



MODULA-2

FOR THE AMIGA

USER MANUAL

DEVELOPER'S VERSION

**SPECIAL FEATURES
LISTED ON BACK**

TDI SOFTWARE, INC.



TDI CONDITIONS OF USE

Copyright and other rights.

TDI programs contain material in which TDI retains proprietary rights. TDI wants these programs to be fully usable by you for the purpose for which they are supplied. No infringement of TDI's rights will occur provided that the following conditions are observed with respect to each program.

1. The programs are used only on a single machine at any one time.
2. The program is copied into machine-readable or printed form only for backup or modification purposes only in support of a single machine.
3. The copyright notice is reproduced and included in any copy or modifications made of the program and in any portion merged into other programs.
4. If this program package is transferred to another party, all copies and modifications made of the program must be transferred or destroyed. You do not retain any right with respect to the transferred package. The other party must agree to observe all the TDI conditions of use.

Any other act involving reproduction of or use of, or other dealing in the programs is prohibited.

No statements contained in this package shall affect the statutory rights of consumers.

TDI Modula-2/Amiga

(c) Copyright 1986 TDI Software Inc.

All rights reserved.

Reproduction or use of editorial or pictorial content in any manner without the express permission of the copyright holder is prohibited.

Whilst every precaution has been taken in the preparation of this manual, no responsibility is assumed for errors or omissions nor is any liability assumed for loss or damage resulting from the use of the information it contains.

Amiga and AmigaDOS are trademarks of Commodore-Amiga Inc.

Published by:

TDI Software Inc.,
10410 Markison Road,
Dallas,
Texas 75238
U.S.A.

Telephone: (214) 340-4942

Telex: 888-442

Contents

<i>Preface</i>	1
Section 1 Introduction	3
1.1 What you have bought	3
1.1.1 The Manual	4
1.2 What you should know about Modula-2	4
1.2.1 Essential Facts	4
1.2.2 Client and Library Modules	5
1.2.3 Program Modules in Modula-2	7
1.3 Access to the Amiga Routines	8
Section 2 Setting up your Modula-2 system	9
2.1 Contents of TDI Modula-2 disks	9
2.2 Installation procedures	9
Section 3 Demonstrations	11
3.1 How to create a working Modula-2 program	11
3.2 How to create a Modula-2 library module	13
Section 4 The Compilation Error Lister	17
4.1 The Compilation Error Lister	17
4.2 Using the Compilation Error Lister	17
Section 5 The Compiler	19
5.1 General Information	19
5.2 Using the Modula-2 Compiler	19
5.3 Embedded Compiler Options	19
5.4 Errors during module compilation	20
5.5 Compiler features and extensions	21
Section 6 The Linker	31
6.1 Description of the Linker's function	31
6.2 How to use the Linker	31
6.3 Module keys and linking	32
Section 7 The Standard Library	33
7.1 Module InOut	33
7.1.1 General Information	33
7.1.2 Definition Module	33
7.1.3 Description of InOut	33
7.2 Module Streams	37
7.2.1 General Information	37
7.2.2 Definition Module	37
7.2.3 Description of Streams	37
7.3 Module String	39

7.3.1	General Information	39
7.3.2	Definition Module	39
7.3.3	Description of String	39
7.4	Module Storage	42
7.4.1	General Information	42
7.4.2	Definition Module	42
7.4.3	Description of Storage	42
7.5	Module SYSTEM	44
7.5.1	General Information	44
7.5.2	Definition Module	44
7.5.3	Description of SYSTEM	44
7.6	Module MathLib0	47
7.6.1	General Information	47
7.6.2	Definition Module	47
7.6.3	Description of MathLib0	47
7.7	Module Terminal	49
7.7.1	General Information	49
7.7.2	Definition Module	49
7.7.3	Description of Terminal	47
Section 8	The Amiga Interface Toolkit	51
8.1	Amiga routines and Modula-2 procedures	51
8.2	Documentation for Amiga routines	51
8.3	Using toolkit routines	52
Appendix A	Amiga demonstration program	55
Appendix B	Modula-2/Amiga Run-time Licence	67
Appendix C	Modula-2 Bibliography	69
Appendix D	Source for all definition modules	71
D.1	ADKBits	72
D.2	Alerts	74
D.3	AmigaUtils	79
D.4	AMIGAX	81
D.5	ANSIConsole	84
D.6	ANSIPrinter	87
D.7	Areas	90
D.8	ASCII	92
D.9	AudioDevice	94
D.10	Blitter	96
D.11	BlitterHardware	98
D.12	CIAHardware	100
D.13	CIAResource	103
D.14	ClipboardDevice	105
D.15	CListLibrary	108
D.16	Colors	114
D.17	CommandLine	116
D.18	ConsoleDevice	118
D.19	ConvToString	121
D.20	Copper	122
D.21	CopperUtils	125
D.22	CustomHardware	127
D.23	Devices	130
D.24	DiskFontLibrary	132

D.25	DMABits	135
D.26	DMAUtils	136
D.27	DOSCodeLoader	137
D.28	DOSExtensions	139
D.29	DOSFiles	144
D.30	DOSLibrary	151
D.31	DOSProcessHandler	154
D.32	Exec	156
D.33	ExecBase	157
D.34	Gadgets	159
D.35	GamePortDevice	161
D.36	Gels	163
D.37	GraphicsBase	170
D.38	GraphicsLibrary	172
D.39	IconLibrary	174
D.30	InOut	177
D.41	InputDevice	179
D.42	InputEvents	180
D.43	IntBits	183
D.44	Interrupts	184
D.45	Intuition	188
D.46	iO	210
D.47	KeyBoardDevice	213
D.48	Layers	214
D.49	Libraries	217
D.50	Lists	220
D.51	MathLib0	223
D.52	Memory	225
D.53	Menus	228
D.54	NarratorDevice	229
D.55	Nodes	232
D.56	ParallelDevice	233
D.57	Pens	235
D.58	PenUtils	238
D.59	Ports	240
D.60	PortUtils	243
D.61	Preferences	244
D.62	PrinterDevice	248
D.63	RandomNumbers	252
D.64	Rasters	253
D.65	Regions	257
D.66	Requesters	259
D.67	Resident	261
D.68	RoundRobinScheduler	263
D.69	Screens	264
D.70	SerialDevice	266
D.71	Sprites	269
D.72	Storage	271
D.73	Streams	273
D.74	String	275
D.75	Tasks	278
D.76	Terminal	283
D.77	Text	285
D.78	TimerDevice	290
D.79	TrackDiskDevice	292

Contents

D.80	TranslatorLibrary	295
D.81	Trapper	297
D.82	Views	298
D.83	Windows	301
D.84	Workbench	304
D.85	Cross reference of definitions	308
Appendix E	Error codes	329
E.1	Run time errors	329
E.2	Compiler error codes	329

Preface

Welcome to TDI Modula-2/Amiga, your sophisticated 32-bit development system for the Commodore Amiga.

TDI Modula-2/Amiga

TDI Modula-2/Amiga is a complete program development environment for Commodore Amiga. It includes:

- * A Modula-2 compiler
- * A linker
- * Library facilities

The TDI Modula-2/Amiga compiler implements the full Modula-2 language as described by Dr Niklaus Wirth in his book "Programming in Modula-2" (Second corrected edition 1983). It includes separate compilation, opaque types, co-routines and floating point routines.

TDI Modula-2/Amiga is a state of the art high level language, simple enough for beginners to pick up easily, and powerful enough for serious programmers to write large complicated programs.

Using TDI Modula-2/Amiga you can create programs which can fully use the advanced features of the Amiga's systems software.

Features of TDI Modula-2/Amiga include:

- * Separate compilation with strong type checking
- * Full support for the Amiga systems routines
- * Compact code

Using this manual

This manual is divided into eight chapters and six appendices. Chapter one helps you find out exactly what you have bought, while chapter two guides you through setting up your Modula-2 system onto the Commodore Amiga. Chapter three takes you through a program and library module creation demonstration.

Chapters four, five and six explain the use of the Compiler and Linker. We do suggest that you read these chapters carefully before trying out your new Modula-2 software.

Chapter seven explains the interface to the Library modules, and chapter eight the interface to the Amiga routines. These chapters are intended to be used more for reference purposes.

The appendices include a wide range of useful information to the Modula-2 programmer. There are sample programs, error messages and some general information. We do recommend that you read through these appendices before beginning programming as they will provide answers to many questions that will arise during the early part of your learning curve.

Chapter 1

Introduction

1.1 What you have bought

You have bought a software development system for the Commodore Amiga microcomputer. You can now develop programs in the programming language Modula-2. The following tools are part of your system:

- * The TDI Modula-2/Amiga Systems Reference Manual
- * A Modula-2 compiler
- * A Program Linker
- * Library facilities

You are reading the manual, everything else mentioned above is on the disks packaged with this manual.

1.1.1 The Manual

This manual includes information on how to use and understand the TDI Modula-2/Amiga system. It also provides information on using all the above-mentioned tools.

However this manual does not provide sufficient information for you to learn the programming language Modula-2, and we recommend that you choose one of the books from the Bibliography to help you.

Also TDI Modula-2/Amiga does not provide full documentation of the Amiga's systems software, these manuals are available from Commodore Amiga Inc. To write any serious software that uses the Amiga's systems software you will need these manuals. The Amiga routines called by the Modula-2 library are not supported, maintained or supplied by TDI. They are proprietary to Commodore Amiga Inc.

High level interfaces to the Amiga routines have been provided so that you can write Amiga programs with your Modula-2, and full details of these can be found in Chapter 8.

1.2 What you should know about Modula-2

1.2.1 Essential Facts

Modula-2 is an advanced high level programming language designed by the creator of Pascal, Professor Niklaus Wirth. In 1983 TDI signed an agreement with Professor Wirth for the commercial rights to the software that he has been developing at ETH in Zurich over the past 5 years. This new Modula-2 compiler has been directly developed from some of the software acquired under this agreement.

Computer hardware has developed rapidly in the past ten years because it has become less difficult to take individual hardware chips such as a microprocessor, some memory, a floppy disk controller and so on, and by using modern CAD techniques, to design a new computer. Software development however has lagged well behind. In the past large and complex software systems were built in such a way that separate parts of the system were always interfering with each other. This caused many development and

maintenance headaches, not to mention weeks and months of debugging.

Modula-2 provides tools to overcome these problems. In order to implement a software system successfully the job must be broken down into smaller manageable tasks. The tool which allows this in Modula-2 is the module.

Every piece of software written using TDI Modula-2/Amiga is a module, and is split into two parts: a definition and an implementation. The definition defines exactly what the module does, which variables and procedures it is importing and what it is exporting. There are no exceptions, no global variations, no chance of slipping in code which will affect other coding in a long forgotten part of the system.

Implementations of modules can be developed, debugged and tested. They then become a "software chip", that is a software module which can be used by a software systems designer to build systems in the same way as a hardware engineer uses hardware chips. Software chips can be used and reused within any appropriate software program. In this way it is comparatively easy for you to design and write large complex systems dealing with big quantities of data, graphic images, speech input systems. Because of the software chip philosophy, the single most crucial aspect which has enabled the explosion of hardware is now available to software writers.

To sum up, the advantages of your new language are:

1. They allow your programming task to be divided up in small, manageable parts.
2. You can alter parts of your programs without affecting other parts.
3. The software modules that you write become part of a growing library of software chips which you can use to build new systems.

1.2.2 Client and Library Modules

There are two types of modules that you can create in Modula-2:

1. Client Modules
2. Library Modules

A Client module is one which "imports" facilities from a separately compiled Library module. These facilities may be constants, types, variables, or

procedures. The Client module then works on the facilities which it has imported, performing the task which it has been designed to do. It does not "export" anything to other modules.

A Library module is one whose basic task is to "export" a specific set of facilities (constants, types etc) to a Client module. Library modules can themselves import facilities from another library module.

Library modules have two parts:

1. A definition module
2. An implementation module

The definition module defines what the Library module does, specifies the tasks it will perform and the facilities which it contracts to export, and is accessible to Client modules.

The implementation module defines how the Library module will do its task, performs the actual tasks and is hidden from Client modules and programs.

The definition part of a module is the interface between the Library module and the Client Modules which it serves. The implementation part does the actual work of the Library module. This separation of the Library module into two parts allows you to change the way in which a task is performed without changing the end results, or affecting the Client modules which import facilities from it.

Definition modules have the following basic pattern:

```
DEFINITION MODULE <the module\'s name> ;  
FROM <library module> IMPORT <list of items> ;  
<declarations of constants, types and variables>  
<procedure declarations>  
END <the module\'s name>.
```

Implementation modules have the following basic pattern:

```
IMPLEMENTATION MODULE <the module's name> ;  
FROM <library module> IMPORT <list of items> ;  
<declarations of constants, types and variables local to  
module>  
<procedure declarations>  
BEGIN  
    <optional initialisation statements> ;  
END <the module's name>.
```

NOTE: The implementation and definition module names must be identical. You should also note that the implementation module cannot export anything not listed in the definition module.

Anything which has been listed inside the implementation module which has not been listed in the definition module is private to the implementation module and cannot be exported. Because of this it should be completely impossible for Client modules to interfere with the operation of carefully written implementation modules.

1.2.3 Program Modules in Modula-2

In Modula-2 programs are always Client Modules. Every program will use facilities from Library modules. Programs generally have one main module and have the following basic pattern:

```
MODULE <the module's name> ;  
FROM <library module> IMPORT <list of items> ;  
<declarations of constants, types and variables local to  
module>  
<procedure declarations>  
BEGIN  
    <statements that perform the programs tasks> ;  
END <the module's name>.
```

Chapter 3 contains more information on how to create your own Modula-2 programs.

1.3 Access to the Amiga routines

Your TDI Modula-2/Amiga system comes complete with a Amiga interface library written in Modula-2. The library has been structured to allow easy access to the AmigaDOS and ROMKernel services.

Chapter 8 contains details of the Amiga Interface Library.

Chapter 2

Setting up your Modula-2 system

2.1 Contents of the TDI Modula-2/Amiga disks

The TDI Modula-2/Amiga system is contained on two Floppy disks. The first contains the Modula-2 compiler, linker, and utilities. The second contains the standard library and the Amiga toolkit.

2.2 Installation procedures

The first thing you should do before attempting to use your Modula-2/Amiga system is to backup the release disks to your own diskettes. For details of how to copy disks on your Amiga, please see page 4-31 of your "Introduction to the Amiga" manual. The release disks should then be stored in a safe place.

Before using your Modula-2 software you will need to understand the concepts used by the AmigaDOS software via the Command Line Interface (CLI). You should read the introduction in the AmigaDOS user's manual to ensure you understand the concepts of AmigaDOS file names, directories, devices etc.

If you have two disk drives on your Amiga you can use the disks as they are supplied, one to hold the compiler etc., the other to hold the modules you are working on and the libraries.

If you only have one disk drive on your Amiga you will have to organise the particular files you require onto one disk. Due to the size of the Amiga's software, we cannot fit all the software onto one disk.

You will require the following items on your main disk:

- 1) The Modula-2 compiler, linker, and any tools you are using (stored under the directory "C").
- 2) The Modula-2 compiler overlays (stored under the directory "L").
- 2) The symbol (.sym) files for any part of the libraries you are using (stored under directory "M2").

3) The link (.lnk) files of all the libraries you are using (stored under directory "M2").

4) The CLI and CLI commands that you will be using.

You can copy the files between disks using the AmigaDOS **Copy** command. Space can be increased by deleting the WorkBench files via the AmigaDOS **Delete** command. Further details of the AmigaDOS commands are contained in the AmigaDOS user's manual.

Before you attempt to use your Modula-2 software you must ensure that the system is setup correctly. This involves assigning the logical devices used by the Modula-2 compiler. The Modula-2 software uses the directory "M2" to store the symbol and link files, you must therefore use the AmigaDOS **Assign** command to tell the compiler which disk this directory resides on. For example if the compiler disk is in the internal disk drive the command:

```
AssignM2 df0:M2
```

will setup the correct logical path. The compiler also uses the general purpose scratch director "T", this must also be assigned to a disk drive or RamDisk before using the compiler or linker.

The simplest way to perform the above steps is to add the relevent commands to the startup file "s:startup-sequence" run when the CLI is invoked. A sample file called "startup-sequence" is included on the first disk to demonstrate setting up the system.

Chapter 3

Demonstrations

3.1 How to create a working Modula-2 program

In this section, you will be shown how to create a simple, working Modula-2 program which can be executed on your Amiga computer. The actual program given will do nothing more than print a message saying Hello on the screen, but even for this small program you will have to use all four major system components:

- 1 The editor, for entering the program text.
- 2 The compiler, for translating the program into machine instructions.
- 3 The standard library, which provides functions to write text onto the screen.
- 4 The linker, for binding the library routines to your program and actually generating an executable program.

This process should familiarise you with the basic operation of the compiler and linker, but a fuller description of all the more advanced features of these products is contained in the following three chapters of this manual, and a full description of the standard library is contained in chapter 7.

However, this process will not teach you how to program in Modula-2, something which is beyond the scope of this manual. If you are intending to write any non-trivial programs, and you are not familiar with the language Modula-2, then you should consult one of the books listed in the Modula-2 bibliography (appendix E).

Before using your TDI Modula-2/Amiga system you should ensure that you have read your "Introduction to the Amiga" manual, and understand the basic manipulation of windows, icons etc.

The Amiga text editor and your TDI Modula-2/Amiga system are both accessed via the Amiga Command Line Interface (CLI). To enter the CLI from the workbench perform the following steps:

- 1) Open the "Preferences" tool, and ensure that the "CLI" option has "YES" selected. Then select the "USE" option and exit the preferences tool.
- 2) Open the "System" drawer.
- 3) Locate the "CLI" icon (a cube with "1" inside it), and open it.
- 4) Select the CLI window.

You are now ready to enter commands to the Amiga system via the CLI interface.

To enter the Amiga's screen editor type "ED Hello.mod" followed by RETURN. For full details of the use of the Amiga text editor please refer to the Commodore AmigaDOS manual.

You are now ready to enter the text of the module. Simply type the text using the BACKSPACE key to correct typing errors. The program text to type in is as follows:

```
MODULE Hello ;  
FROM Terminal IMPORT WriteString, WriteLn;  
  
BEGIN  
  WriteString("Hello from a Modula-2 Program") ;  
  WriteLn ;  
END Hello.
```

Be careful to type all the letters in the correct case (Modula-2 keywords must all be in capitals), or the program will not compile correctly, and you will have to re-edit it to remove the mistakes.

To save the file type the ESC key to enter command mode, and type "X" followed by RETURN.

The editor will save the file and return you to the Amiga command line.

The next stage in the process is compiling the program you have just typed in.

To invoke the Modula-2 compiler enter the command "modula". The compiler should load and issue a prompt telling you which version it is. If the compiler

does not start correctly, check you have installed the release disk correctly as described in chapter two.

In response to the prompt for the file to compile, enter "Hello". The compiler will then compile the module text you have just typed in.

If the program was not compiled successfully, the compiler will terminate saying that errors occurred during compilation, and it will write out a file with error messages on it and you will have to re-edit the file.

If all was well, then skip the rest of this paragraph, and move onto the next one. If an error occurred, you must re-edit the program to correct the appropriate part of the text. When the compiler detected the compilation errors, it will have started another task running the compilation error lister utility. This utility lets you scan through the errors while you correct the source of your module. If you do have compilation errors please read chapter 4 now, which describes the error lister. Once the file has been corrected it can then be saved on the disk, as explained above. You can then attempt to re-compile the program.

Having successfully compiled your first program, the final stage in the process is to link it, that is bind your program code to the code from the imported modules, giving a file which can be executed directly.

To link your module, invoke the Modula-2 linker by entering the command "link". When prompted for the file to link enter "Hello". The linker will then search for the modules needed to produce the program.

When the linker has finished, you will be returned to the Amiga CLI. To run your program simply enter the command "Hello". The program should load, issue the hello message, and return to the CLI.

3.2 How to create a Modula-2 library module

The module created in the previous section is a stand-alone program. In this section a simple library module will be developed which generates pseudo-random numbers. For a library module there will be both a definition and an implementation module. Both have to be typed into the editor and compiled with the compiler.

The text of the definition module is given below. Type this into the editor and call it RandomNumbers.def.

```
DEFINITION MODULE RandomNumbers;  
  
PROCEDURE Random ( MaxValue : CARDINAL ) : CARDINAL;  
(* Return a random number in the range 0..MaxValue *)  
  
END RandomNumbers.
```

The text of the implementation module is given below. Type this into the editor and call it RandomNumbers.mod.

```
IMPLEMENTATION MODULE RandomNumbers;  
CONST M = 1000000000;  
      m1 = 10000;  
      b  = 31415821;  
VAR seed : LONGCARD;  
  
PROCEDURE Random ( MaxValue : LONGCARD ) : LONGCARD;  
  
  PROCEDURE Multiply ( p, q : LONGCARD ) : LONGCARD;  
  VAR p0, p1, q0, q1 : LONGCARD;  
  BEGIN  
    p1 := p DIV m1;  
    p0 := p MOD m1;  
    q1 := q DIV m1;  
    q0 := q MOD m1;  
    RETURN (((p0*q1+p1*q0) MOD m1) * m1 + p0*q0) MOD M;  
  END Multiply;  
  
BEGIN  
  seed := (Multiply (seed, b) + 1) MOD M;
```

```
RETURN ((seed DIV m1) * MaxValue) DIV m1;  
END Random;
```

```
END RandomNumbers.
```

When both modules have been entered, compile them both, but remember to compile the **DEFINITION** module first. When the definition module is compiled, the compiler generates a file with the suffix **.SYM** ("**SYM**" is short for symbol). This file is needed to compile any modules which wish to use the random numbers module. When the implementation module is compiled, the compiler generates a file with the suffix **.LNK** ("**LNK**" is short for link). This file is needed to bind the random number generator to any modules which use it.

This page intentionally left blank.

Chapter 4

The Compilation Error Lister

4.1 The Compilation Error Lister

The compilation error lister is a utility that allows you to step through the compilation errors of a module, whilst you are correcting the errors in the module's source.

When the compiler encounters errors in a module, it writes an error data file which indicates where the errors occurred in the source. If a listing file is being produced, it also writes error numbers to this file. The error file is created with the extension ".erd" for definition modules, and ".erm" for implementation or program modules. The error lister uses this file in conjunction with the file "Syntax.ind", which contains the text of the error messages, and your source file to indicate where compilation errors occurred.

4.2 Using the Compilation Error Lister

The utility is contained in the file "M2Error". It can be invoked by various means:

- 1) Automatically by the compiler when compilation errors are found.
- 2) By typing "run M2Error" to create a new task under the Amiga CLI.
- 3) By typing "M2Error" under the Amiga CLI.

The first two options create a separate task which runs the error lister. This is desirable because this allows you to run the text editor from the first CLI, and swap between the error lister and editor whilst correcting your source. Using the error lister from the normal CLI is useful for general scanning of errors.

The template for the error lister command is:

M2Error <name>

where <name> is the filename of the source file for your module.

When started the error lister opens both your source file and the error file, and indicates how many compilation errors were found. For each error in turn, the source line in error, and the previous two lines of source are displayed together with an indication of the position and type of error. If the error lister could access the syntax file "Syntax.ind", error messages will be presented in text form, if this file could not be found the error numbers will be displayed. A full list of the error numbers and their messages are contained in appendix F.

After each error is displayed, you have the choice of entering "Q" to finish scanning for errors, or any other character to continue.

Chapter 5

The Compiler

5.1 General information

The TDI Modula-2/Amiga system includes a multi-pass compiler for the language Modula-2. It is a full implementation of the language together with some extensions to the language as specified in the book "Programming in Modula-2" by Prof. Niklaus Wirth.

The compiler takes as input text files containing either the definition part of a module, or the implementation part. As output the compiler generates symbolic data for definition modules and relocatable 68000 native-code link files for implementation modules.

5.2 Using the Modula-2 compiler

The modula-2 compiler is accessed via the Amiga's Command Line Interface (CLI). The CLI and its commands are fully described in the Commodore AmigaDOS manual.

The template for the Modula-2 command is:

MODULA [<name>] [LIST] [QUERY]

the parameters are defined as follows:

<name> = Filename to compile. If not specified no options may be entered and the filename will be prompted for.

LIST = Specify that a listing file should be produced as "<name>.LST".

QUERY = When searching for imported modules this option causes each module's filename to be prompted for.

5.3 Embedded Compiler Options.

Comments in a Modula-2 compilation unit may be used to specify certain compilation options which affect the object code produced.

A compilation option is made up of the character "\$" followed by one of the (uppercase) letters "C", "P", "S", "T", followed by one of the characters "+", "-", "=".

The options have the following effect:

C : Generate the normal 68000 clear instructions to set variables to zero. The only time this is not required is if you are trying to clear a register on a device and the read before write operation of the clear operation will cause a problem.

P : Generate the normal entry and exit code for the next procedure. The only time this is not required is if a procedure is being jumped to from a machine code instruction.

S : Generate code at the entry of each procedure to check the stack space available.

T : Generate code to perform range checking on array subscripts etc.

The switches have the following effect:

+ : The option is enabled, this is the default setting.

- : The option is disabled.

= : The option is restored to its former state.

Compilation options must be the first options within a comment. An option is not recognised by the compiler if any information (including blank spaces) is in front of it. Examples of options which will be recognised by the compiler are:

(***\$T-** turn off range checking *)

(***\$T+** turn it back on *)

(***\$T=** turn it back off *)

5.4 Errors during module compilation

Errors may occur during compilation for two major reasons:

- 1 The text of the module being compiled is not a correct Modula-2 program.
- 2 The compiler could not find or access one of the files which it needed to reference in order to compile your module.

If the error is with the program source, the compiler will inform you of this, and will write a file out to the disk which lets the error lister know exactly where in the module the errors occurred, and also what sorts of errors occurred. All you have to do is re-edit the program to remove the errors, and then attempt to re-compile the program. A full description of the workings of the editor is contained in your AmigaDOS users manual. A list of the compiler's error messages is contained in Appendix F.

If the compiler cannot find a file which it is looking for, it will give the name of the file it was looking for and will prompt you to enter a new file, or continue. This sort of error happens most often when the compiler is looking for a symbol file of one of the modules which is being imported into your program. If you wish to avoid these errors then you should always use the first 8 letters of the module's name as the first 32 letters of the filename.

5.5 Compiler features and extensions.

Extensions to the Modula-2 language

These extensions to the language conform to those outlined by Prof. N. Wirth in his paper "Revisions and extensions to Modula-2. 1.2.84/14.5.84"

Definition module EXPORT list

All objects declared in a definition module are now automatically exported. The explicit export list may be discarded. If it is specified, it will be ignored.

CASE statement syntax change

The syntax of the CASE statement and its variant record declaration has been changed from

```
case = CaseLabelList ":" StatementSequence
variant = CaseLabelList ":" FieldListSequence
to
case = {CaseLabelList ":" StatementSequence}
variant = {CaseLabelList ":" FieldListSequence}
```

This change allows the inclusion of an empty case with a trailing bar (|) character.

Example:

In the old syntax the statement

```
CASE char OF  
  'A' : DoACommand; |  
  'B' : DoBCommand; |  
  'C' : DoCCommand;    (* no | here *)  
END;
```

could confuse people due to the last CASE label not being allowed to have the bar (|) character included. The new syntax allows:

```
CASE char OF  
  'A' : DoACommand; |  
  'B' : DoBCommand; |  
  'C' : DoCCommand; | (* | allowed *)  
END;
```

N.B. Please also note that the new syntax allows a totally empty CASE statement of the form

```
CASE char OF  
END;
```

one should therefore not be caught out when this causes a "CASE label range error" if range checking is turned ON, as the expression char does not match any of the non-existent CASE labels.

String and CHAR compatibility

A string of (n) characters is said to have a length of (n). The new compiler allows a string of length 1 to be assignment compatible with the type CHAR.

Example.

The following code is now supported:

```

VAR      s : ARRAY [0..10] OF CHAR;
BEGIN
    s := "A";  (* "A" is a CHAR CONST *)
END;

```

Synonym for NOT

The character "~" (176C) is now a synonym for the not symbol.

Long simple types

The new identifiers LONGCARD and LONGINT denote standard types. These types are the 32-bit wide counterparts to the standard types CARDINAL and INTEGER. All standard operations are supported on these types. It should be noted that they are not compatible with the standard 16-bit types, although this restriction can be circumvented by the use of type transfer functions.

Implementation note : It should be noted that due to the structure of the Motorola 68000 processor, the operators "*" and "DIV" will slower on the long types than on the standard 16-bit types.

Example.

The following is legal code :

```

VAR long  : LONGCARD;
    short : CARDINAL;
BEGIN
    long := 31415927;
    short := CARDINAL (long MOD 6666);
END;

```

Implementation dependent extensions

The following enhancements to the Modula compiler have been developed in versions of the compiler greater than 2.00a. Please note that these features may not be available on other implementations of the language, and care should therefore be taken when transporting modules to other

implementations.

New formal parameter types

The module SYSTEM now exports the formal parameter types BYTE and LONGWORD to complement the existing WORD type.

If a parameter of a procedure is given the type byte then the corresponding actual parameter may be of any type that occupies one byte (8 bits).

The same applies for parameters of type WORD and LONGWORD, except that the relevant sizes in bits are 16 and 32, respectively.

Example.

```
FROM SYSTEM IMPORT BYTE, WORD, LONGWORD,
                    ADDRESS;

PROCEDURE TakeAByte ( VAR b : BYTE );
PROCEDURE TakeAWord ( VAR w : WORD );
PROCEDURE TakeALong ( VAR l : LONGWORD );

VAR
    ch      : CHAR;
    card    : CARDINAL;
    int     : INTEGER;
    lcard   : LONGCARD;
    addr    : ADDRESS;
BEGIN
    TakeAByte (ch);
    TakeAWord (card);
    TakeAWord (int);
    TakeALong (lcard);
    TakeALong (addr);
END;
```

Boolean constant expression optimisation

The processing for the IF and WHILE statements has been extended to

optimise the statement if the result of the boolean expression is a constant.
Example.

The IF statement optimises the generation of code for expressions.

In the following program fragment :

```
CONST
    Debug = FALSE;
BEGIN
    .
    .
    IF Debug THEN
        j := 0;
        FOR i := 0 TO 1000 DO
            INC (j, i);
        END;
    END;
```

The compiler generate no code for the IF statement because the expression evaluates to the constant FALSE. From the example, it can be seen that this can be very useful when debugging code is required during a modules development, but can then be quickly disabled, causing no overhead in the finished module.

It should also be noted that sub-expressions containing boolean constants are also optimised out of the code generation.

Example:

given the declarations,

```
CONST
    t = TRUE;
    f = FALSE;
```

in the expressions :

```
(x * y DIV z > a) AND f
(x * y DIV z > z) OR t
```

the entire $(x * y \text{ DIV } z) a$ will be optimised out in each case, as it cannot affect the result of the boolean expression.

N.B. It should be noted that function procedures generating "side-effects" as noted in N. Wirth's book "Programming in Modula-2" (p.52) may produce unexpected results due to their not being called at all. It should be stressed that it is bad programming practise to modify non-local data in function procedures.

Example.

```
IF (function (42) = 1) AND FALSE THEN ... END;
```

in the above statement, the function procedure is never called.

The SET type

The compiler includes many new features to support the SET type. The most major of these is the implementation of large sets. The compiler optimises the length of a set according to its maximum element. The following table indicates the set sizes generated:

SET range	size in bytes
0..7	1
0..15	2 (word)
BITSET	2 (word)
0..31	4 (longword)
0..n (n > 31)	$(n + 7) \text{ DIV } 8$

The maximum allowed number of elements in a set supported by the compiler is 65,536 (64k). A set of the maximum size therefore requires 8k bytes of storage.

It should be noted that large sets incur an overhead in processing due to the structure of the 68000 processor.

SET OF CHAR

The compiler relaxes the Modula-2 syntax to allow the specification of the type SET OF CHAR. This type has an element range of 0..255, requiring 32 bytes of storage.

Examples.

TYPE

```
chset = SET OF CHAR;
```

CONST

```
alphabet = chset {"A".. "Z", "a".. "z"};
```

VAR

```
ch : CHAR;
```

```
cset : chset;
```

```
IF ch IN alphabet THEN
```

```
  .
```

```
  .
```

```
  .
```

```
END;
```

```
cset := chset {};
```

```
INCL (cset, ch);
```

```
IF "a" IN cset THEN
```

```
  .
```

```
  .
```

```
  .
```

```
END;
```

Large sets

The code generation for large sets is optimised to reduce the overhead of processing these types. The following notes will enable the user to reduce the processing time of these types even further.

a) Set constants.

It should be noted that set constants have to be allocated in the code space of the program. The exception to this is the empty set. The use of an empty set incurs no data space overhead.

Example.

TYPE

```
lset = SET OF [0..1023]; (* 128 bytes *)
```

```
s := lset{0..1023}; (* allocates 128 bytes*)
```

```
s := lset{}; (* no data allocation *)
```

b) Set expressions.

Large set expressions are optimised well by the compiler. It is therefore better to use a single set expression rather than multiple statements.

Example.

Instead of:

```
(c IN set1) OR (c IN set2) OR (c IN set3)
```

Write:

```
(c IN (set1 + set2 + set3))
```

5.6 Modula-2/Amiga specific compiler notes

This section itemises the implementation specific differences and restrictions for the Amiga Modula-2 compiler.

Open array parameters

Open array parameters (eg. ARRAY OF CHAR) must be specified as VAR parameters, but may be called by values in the case of string constants.

Example.

```
PROCEDURE WriteString (VAR s: ARRAY OF CHAR);
```

```
WriteString ("This is a string constant.");
```

FOR Statement

The FOR statement control variable must not be of byte size if the step increment is not -1 or 1.

The step size in FOR statements must not be greater than 32767.

CASE statement

The labels of a CASE statement must not have any values greater than 32767.

Set expressions as parameters

Set expressions used as parameters must not be large sets. Large sets are those with greater than 32 elements.

This page intentionally left blank

Chapter 6

The Linker

6.1 Description of the Linker's function

The linker is needed to convert the files the compiler produces into files which can be recognised as executable code files by the Amiga disk operating system. The linker is also vitally important, as it is responsible for binding together all of the modules that a program imports to the program, so that these imported routines are available at runtime. Even a program which imports no modules at all must be put through the linker so that the small amount of runtime support code which is needed to execute a program can be stitched into it (this runtime code is contained in the vitally important **AMIGAX.LNK** file which is supplied on one of your disks).

6.2 How to use the Linker

The modula-2 linker is accessed via the Amiga's Command Line Interface (CLI). The CLI and its commands are fully described in the Commodore AmigaDOS manual.

The template for the modula-2 linker command is:

LINK [<name>] [QUERY] [MAP [=<name>]]

the parameters are defined as follows:

<name> = Filename to link. If not specified no options may be entered and the filename will be prompted for.

QUERY = When searching for imported modules this option causes each module's filename to be prompted for.

MAP = Specifies that a load map listing should be produced. This is either written to the name supplied, or the main name with the extension ".MAP".

When searching for imported modules, the linker prompts for files that it cannot find. If the file which the linker could not find has a different name from that which the linker was looking for, then simply enter the filename that the linker should be looking for, otherwise enter nothing and the linker will see if

the other files needed are available, but will not generate an executable program file.

The output of a successful linkage is an executable program with the the name of the main program module. This file can be executed by typing its name under the Amiga command line.

6.3 Module keys and linking

When a definition module is compiled, the compiler generates a so-called module key. This key is unique, and is needed to distinguish between different versions of the same module. This key is written into the symbol file of the definition module. When an implementation module is compiled, the compiler assigns to it the module key it finds in the symbol file of the corresponding definition module, and places this key in the link file it generates. This is the systems way of insuring compatibility between definition and implementation modules. If a definition module is recompiled (even if it is unchanged), a new module key will be generated, and all other modules which import the module will have to be recompiled, along with the module s implementation. Incompatible module keys will cause errors at link time, and the solution to these is to recompile all the modules which import the changed modules. Recompilation of an implementation module will not affect the module key.

Chapter 7

The Standard Library.

7.1 Module InOut

7.1.1 General Information

InOut is the standard module used for reading and writing numbers and text strings to and from the screen, keyboard and disk files. InOut is a module which should be available on all Modula-2 implementations, and so using this module should make programs portable to other Modula-2 systems. The input and output sources for InOut default to the keyboard and the screen respectively. These defaults can be changed by calling **OpenInput** and **OpenOutput**. If input and output is subsequently wanted from the keyboard and the screen, **OpenOutput** and **OpenInput** can be called with the filename set to "con:".

7.1.2 Definition module

A full listing of the InOut definition module is contained in appendix D.1

7.1.3 Description of InOut

CONST EOL

This constant represents the character used to mark the end of a line. Thus, a call to **WriteLn** (below) is equivalent to **Write (EOL)**

VAR In, Out

These variables represent the current input and output streams.

VAR Done

This variable is used to indicate the success or failure of an InOut operation. Done is set after every InOut operation. If it is **TRUE** the operation succeeded, if **FALSE** it failed.

VAR termCH

This is used as the terminating character in calls to ReadInt and ReadCard.

OpenInput (VAR defext : ARRAY OF CHAR)

This procedure asks for an input filename. defext contains the default file extension.

OpenInputFile(VAR FileName : ARRAY OF CHAR);

This procedure attempts to open the file specified in FileName for input, and assign the stream in to it.

OpenOutput (VAR defext : ARRAY OF CHAR);

This procedure asks for an output filename. defext contains the default file extension.

OpenOutputFile(VAR FileName : ARRAY OF CHAR);

This procedure attempts to open the file specified in FileName for output, and to assign the stream out to it.

CloseInput;

This procedure closes the current input file and returns the input to the keyboard.

CloseOutput;

This procedure closes the current output file and returns the output to the screen.

Read (VAR ch : CHAR);

This procedure attempts to read a single character from the current input stream. If the input stream is a file, then Done becomes FALSE if the end of the file was hit.

ReadString (VAR s : ARRAY OF CHAR);

This procedure reads a sequence of characters (not containing blanks or control characters) from the current input stream. Input is terminated by any character with an ASCII value of less than 33. This character is assigned to termCH. If input is from the terminal then ASCII.DEL (value=127) can be used for backspacing.

ReadInt (VAR x : INTEGER);

This procedure reads an integer from the input stream. Leading blanks are ignored. When a character not equal to a digit "0".."9" is encountered, input terminates.

ReadCard (VAR x : CARDINAL);

This procedure reads a cardinal number in the same way as ReadInt reads an integer.

Write (ch : CHAR);

This procedure writes the character passed to the current output stream.

WriteLn;

This procedure terminates a line by writing a carriage return and then a linefeed character.

WriteString (VAR s : ARRAY OF CHAR);

This procedure writes the string passed to the current output stream. A string is a sequence of characters terminated by an ASCII NUL (0C).

WriteInt (x : INTEGER; n : CARDINAL);

This procedure writes an integer number to the stream out. n is greater than the number of digits, blanks are written preceding the number, so that the total number of characters written is n.

```
WriteCard ( x : CARDINAL );  
WriteOct  ( x : CARDINAL );  
WriteHex  ( x : CARDINAL );
```

These three procedures work in the same way as WriteInt except that they write cardinal, octal and hexadecimal numbers respectively.

7.2 Module Streams

7.2.1 General Information

Streams is a module which is used for general file handling. It offers random access facilities (in contrast to InOut which is purely sequential).

7.2.2 Definition module

A full listing of the Streams definition module is contained in appendix D.1

7.2.3 Description of Streams

TYPE Stream

When Connect is called, a variable of type Stream has to be passed to the procedure. This variable should be subsequently used in all calls to procedures in Streams.

Connect (VAR s: Stream ; f: FileHandle)

This procedure takes a file handle returned by the module DOSFiles and connects it to the stream type "s".

Disconnect (VAR s: Stream)

This procedure performs the reverse of Connect.

WriteChar (s : Stream; ch : CHAR)

WriteWord (s : Stream; w : WORD)

These three procedures write the quantities passed to the stream indicated.

ReadChar (s : Stream; VAR ch : CHAR)

ReadWord (s : Stream; VAR w : WORD)

These three procedures read the quantities specified from the stream indicated.

Reset (s : Stream)

This procedure positions the file at its beginning.

GetPos (s: Stream; VAR position : LONGCARD)

SetPos (s: Stream; position : LONGCARD)

These two procedures get and set the stream position pointer respectively. The pointer is the offset of the stream pointer into the stream in bytes.

7.3 Module String

7.3.1 General Information

Strings provides elementary string handling routines which can be easily accessed by user modules. The procedures in this module operate on arrays of characters, and these arrays may be of any length.

7.3.2 Definition module

A full listing of the Strings definition module is contained in appendix D.3

7.3.3 Description of String module

Strings exports eleven procedures, two data types, and one constant. A description of the function of each of these objects is given below:

CONST MaxChars

This constant is an arbitrary one, set to 80, which suggests a maximum length for strings. However, Strings will operate on a string of any length. For normal string usage, ie writing messages to the screen etc., MaxChars will be a convenient maximum string length.

TYPE String

This type is a string of length MaxChars. If a longer (or shorter) string is needed, then the user can simply define a new string type.

TYPE CompareResults

This type is used by the procedure which compares two strings. Compare always returns one of the three results Greater, Equal or Less.

InitStringModule;

This procedure simply initialises the strings module, putting it into its startup state.

```
Assign ( VAR Dest   : ARRAY OF CHAR;  
         VAR Source  : ARRAY OF CHAR );
```

This procedure allows strings to be assigned to each other in one operation. It is equivalent to the operation `Dest := Source`.

```
Insert ( VAR SubStr : ARRAY OF CHAR;  
         VAR Str     : ARRAY OF CHAR;  
         Index       : CARDINAL );
```

This procedure inserts the string `SubStr` into the string `Str` at the position indicated by `index`.

```
Delete ( VAR Str    : ARRAY OF CHAR;  
         Index       : CARDINAL;  
         Len         : CARDINAL );
```

This procedure removes `Len` characters from the string `Str` starting at the character specified by `Index`.

```
Copy ( VAR Str      : ARRAY OF CHAR;  
       Index         : CARDINAL;  
       Len           : CARDINAL;  
       VAR Result    : ARRAY OF CHAR );
```

This procedure copies `Len` characters from the string `Str`, starting at the character specified by `Index` into the string `Result`.

```
Concat ( VAR s1, s2 : ARRAY OF CHAR;  
         VAR Result  : ARRAY OF CHAR );
```

This procedure adds string `s2` to string `s1` and places the resulting string into `Result`. It is equivalent to `Result := s1 + s2`.

```
Length ( VAR Str : ARRAY OF CHAR ): CARDINAL;
```

This function procedure returns the length in characters of the string

passed to it.

```
Compare ( VAR s1, s2 : ARRAY OF CHAR )  
          : CompareResults;
```

This procedure compares the two strings passed to it on a character by character basis. It uses the ASCII value of the characters to determine which is greater than the other. The procedure will order strings in the same way as a dictionary orders its entries, but several special cases can occur : If both strings are the same length and have the same content, then the procedure returns Equal. If both strings are identical up to the end of one string, then if s1 is longer than s2 the procedure returns Greater, otherwise it returns Less.

```
Pos ( VAR Source : ARRAY OF CHAR;  
      VAR Match  : ARRAY OF CHAR;  
      Start      : CARDINAL;  
      VAR Where   : CARDINAL );
```

This procedure attempts to find the string Match in the string Source, starting at the character specified by Start. If a match is found, it returns the position of the start of the match in Where.

```
SetTerminator ( Ch : CHAR );
```

This procedure sets the character that module Strings uses to mark the end of a string. The default value is OC (CHR (0)).

```
GetTerminator () : CHAR;
```

This procedure returns the character that the strings module is currently using to mark the end of strings.

7.4 Module Storage

7.4.1 General Information

The module Storage contains routines to allocate items on a "Heap". A heap is an area of memory set aside for the creation of dynamic structures. The routines **ALLOCATE** and **DEALLOCATE** are mapped by the compiler to the standard dynamic storage routines **NEW** and **DISPOSE**. Before using **NEW** or **DISPOSE** the routines **ALLOCATE** and **DEALLOCATE** must be imported from Storage.

7.4.2 Definition Module

A full listing of the Storage definition module is contained in appendix D.4

7.4.3 Description of Storage

ALLOCATE (VAR Addr: ADDRESS; Amount :CARDINAL);

Allocates Amount bytes of storage on the heap and returns the address in Addr. If the requested amount is too large **ALLOCATE** returns NIL is the GiveNIL option was used to create the heap, or HALTs the program.

DEALLOCATE (Addr: ADDRESS; Amount: CARDINAL);

Deallocates the heap storage reserved by **ALLOCATE**.

CreateHeap (Amount : LONGCARD) : BOOLEAN;

Reserves the area of memory to be used as the heap. **CreateHeap** returns TRUE if the heap was created successfully.

HeapLeft () : LONGCARD ;

This procedure returns the total amount of free memory in the heap.

DestroyHeap ;

This procedure returns the heap storage to the system.

7.5 Module SYSTEM

7.5.1 General Information

The module **SYSTEM** contains system specific features for an implementation. It contains tools to aid low-level programming in Modula-2. It should be noted that any object imported from the module **SYSTEM** may cause transportation of the module to another Modula-2 environment to be difficult.

7.5.2 Definition Module

The module **SYSTEM** is not like any other module in the TDI Modula-2/Amiga package. It has no real definition or implementation part. Paradoxically, though, objects can be imported from **SYSTEM**. This is because **SYSTEM** is resident in the Modula-2 compiler, and so the compiler automatically knows about the types and procedures in **SYSTEM**. However, it is possible to explain the module **SYSTEM** using conventional Modula-2 language, and this is done below.

7.5.3 Description of **SYSTEM** module

TYPE **BYTE**
 WORD
 LONGWORD

These types offer the representation of the basic storage units accessed by the 68000 processor. Each type may substituted in a parameter declaration for any type that exactly represented by the accessed storage unit. No operations are allowed on these types without type coercion.

TYPE **ADDRESS**

The type **ADDRESS** is compatible with all pointer types. It is defined as the type **POINTER TO WORD**. All integer arithmetic operations may be applied to this type.

TYPE **PROCESS**

This type is used for process handling.

```
NEWPROCESS ( processProc : PROC;  
              workspace  : ADDRESS;  
              worksize   : CARDINAL;  
              VAR process : PROCESS );
```

Creates a process as specified by the Modula-2 programming report page 162.

```
TRANSFER ( VAR p1, p2 : PROCESS );
```

Transfers control between processes.

```
IOTRANSFER ( VAR p1, p2 : PROCESS;  
              device : ADDRESS );
```

Transfer procedure for processes operating on a peripheral device.

```
LISTEN
```

Allows interrupts to be serviced. The priority is set to zero.

```
SYSRESET
```

Procedure to initialise the system.

```
CODE ( w : CARDINAL ..... );
```

Allows the insertion of machine code into the object code of the compiling module. Each parameter *w* represents one 16-bit instruction to be executed. The procedure has an arbitrary number of parameters separated by the character “,”.

```
SETREG ( regNum : CARDINAL; value : ADDRESS );
```

Procedure to generate code to set a value into one of the 68000

processor's general purpose registers. regNum is the register number:

D0 = 0 .. D7 = 7, A0 = 8 .. A7 = 15

REGISTER (regNum : CARDINAL) : ADDRESS;

Procedure to generate code to return the contents of one of the 68000 processor's general purpose registers (see SETREG).

ADR (variable) : ADDRESS;

Returns the storage address of a variable.

**SIZE (variable) : CARDINAL / INTEGER /
ADDRESS;**

Returns the number of bytes allocated for a variable. If the variable is a record type with variants, the maximum size is returned

**TSIZE (type) : CARDINAL/INTEGER/ADDRESS;
TSIZE (type, tag1const ...)
: CARDINAL/INTEGER/ADDRESS;**

Returns the number of bytes that will be allocated for the specified type. If the type is a record with variants then the tag constants specify the field variants. If variants are not specified then the maximum size for that variant is returned.

7.6 Module MathLib0

7.6.1 General Information

The module MathLib0 provides all the basic trigonometrical functions, and also functions for converting degrees to radians, and vice-versa. Functions for logarithms and exponentiation are also included. The maths functions are accurate to the seventh digit. The eighth digit is dubious as a 23 bit fractional mantissa is used in the system. Where applicable, procedure names correspond with those proposed by Prof. N. Wirth., and so programs which use this module should be easily transportable to other Modula-2 environments.

7.6.2 Definition module

A full listing of the MathLib0 definition module is given in appendix D.5

7.6.3 Description the MathLib0 module

```
CONST pi  
CONST e
```

These two constants represent Pi and Euler's constant respectively.

```
RadToDeg ( RadianAngle : REAL ) : REAL;  
DegToRad ( DegreeAngle : REAL ) : REAL;
```

These two procedures are used for converting an angle in radians to an angle in degrees, and an angle in degrees to an angle in radians respectively.

```
Real ( x : INTEGER ) : REAL;  
entier ( x : REAL ) : INTEGER;
```

These two procedures are used for converting a real number into an integer, and an integer into a real number respectively.

```
sin ( x : REAL ) : REAL;  
cos ( x : REAL ) : REAL;
```

```
tan ( x : REAL ) : REAL;  
atan ( x : REAL ) : REAL;
```

These four procedures provide the standard trigonometric functions. The parameters to the sin, cos and tan procedures are all taken to be angles in radians, and the procedure arctan returns a number which is to be interpreted as an angle in radians.

```
exp ( x : REAL ) : REAL;
```

This procedure evaluates the exponential of x.

```
ln ( x : REAL ) : REAL;  
log ( x : REAL ) : REAL;
```

These two procedures evaluate the natural logarithm of x and the logarithm of x to the base 10 respectively.

```
power ( x, y : REAL ) : REAL;
```

This procedure evaluates x raised to the yth power.

```
sqrt ( x : REAL ) : REAL;
```

This procedure evaluates the square root of x.

IF the functions arcsin and arccos are needed, they can be calculated using the following formulae:

$$\arcsin(x) = \arctan(x / \sqrt{1-x^2})$$
$$\arccos(x) = \pi / 2 - \arcsin(x)$$

7.7 Module Terminal

7.7.1 General Information

The module Terminal contains procedures to read from the keyboard and write to the screen. These procedures are very primitive, being mainly character orientated. However, they are very general in nature, and can therefore form the base for much more sophisticated input and output procedures.

7.7.2 Definition module

A full listing of the Terminal definition module is given in appendix D.7

7.7.3 Description of objects exported from the module Terminal

Read (VAR ch : CHAR);

This procedure waits until a key is typed at the keyboard and returns the key pressed in the variable ch.

BusyRead (VAR ch : CHAR);

This procedure reads a key from the keyboard if one is waiting. If there is no key waiting to be read, then the procedure returns immediately with the variable ch set to OC (CHR(0)).

ReadAgain;

This procedure causes the last character read from the keyboard to be read again upon the next call to Read.

Write (ch : CHAR);

This procedure writes the character passed in ch onto the screen.

WriteLn;

This procedure writes a carriage return and line feed to the screen.

WriteString (VAR s : ARRAY OF CHAR);

This procedure writes all the characters in the array `s` onto the screen. It does not write a carriage return or a line feed afterwards.

Chapter 8

The Amiga Interface Toolkit

8.1 Amiga routines and Modula-2 procedures

The Amiga is a complex machine with many advanced features. In its raw state, the machine is very awkward to program, and it is for this reason that the Amiga operating system is used on the machine. AmigaDOS provides an easy way of accessing all the advanced features present on the Amiga computer, without the need to write tricky assembler routines. Your Modula-2 system allows full access to all of the Amiga system routines which handle the disk drives and the graphics screen, via high-level Modula-2 procedure calls to a pre-written library of Amiga routines. In this way it is possible to write programs using, for example, the graphics functions entirely in Modula-2.

The Amiga provides two main sets of routines:

- 1 The AmigaDOS ('D'isk 'O'perating 'S'ytem) routines
- 2 The Amiga Kernel routines

All routines are available from Modula-2, but due to the (at first bewilderingly) large number of these routines, they have been divided up into smaller chunks of routines which form logical groupings. All the routines have full and clear names, which should be an aid both to programming and to debugging modules which use the Amiga library.

8.2 Documentation for Amiga routines.

It is beyond the scope of this manual to fully document the Amiga's systems software. You will need the full Amiga manuals if you wish to develop any serious software that uses the Amiga's systems software. Your local Commodore dealer should be able to provide you with the following Commodore manuals. These manuals contain a full description of the systems software on the Amiga.

AmigaDOS User's Manual (327264-01)
AmigaDOS Technical Reference Manual (327266-01)
AmigaDOS Developers Manual (327265-01)
Intuition, The Amiga User Interface (327267-01)
ROM Kernel Manual (327271-01)

8.3 Using toolkit routines

The Amiga ROM kernel manages the systems software as Libraries. A library is a collection of system routines and is accessed from your Modula-2 toolkit. Before you can call the Amiga library routines you must ensure that the library is available for access. To achieve this you must open the library before calling any of the routines. Each main Modula-2 toolkit module contains a variable of the form "`<name>Base : ADDRESS`", where `<name>` is the Amiga library name. To open a library you must call the routine **OpenLibrary** from the module **Libraries**, which returns the pointer to be placed in the base variable. You can then access the library routines. Each module also exports a constant that is the library name for the **OpenLibrary** call. Before your module terminates it must also close all of the libraries that it opened. This is performed by the **CloseLibrary** function. The following code fragment shows how to use an Amiga library :

```
FROM Libraries IMPORT OpenLibrary,
                        CloseLibrary ;
FROM Intuition IMPORT IntuitionBase ,
                        IntuitionName ;
FROM Windows    IMPORT WBenchToFront ;

IntuitionBase := OpenLibrary(IntuitionName,0);
IF WBenchToFront() THEN
    .
    .
END ;
CloseLibrary(IntuitionBase) ;
```

For further details of libraries and their use please see Chapter 5 of the

Amiga ROM Kernel manual.

To use any of the toolkit routines, one must first import them into the client module. The procedures can then be called the same way as any other Modula-2 procedure. Appendix A contains an example program illustrating the use of procedures from the Amiga toolkit library. An executable version of this program is also supplied on your release disks.

This page intentionally left blank

Appendix A

A.1 Amiga demonstration program.

This appendix contains a listing of an Amiga program written in Modula-2. The program displays a rotating, shaded cube on the Amiga screen. It is included to act as a guide to using some portions of the Amiga toolkit modules.

MODULE Cube;

(** -----
Commodore Amiga rotating solid cube demonstration module
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved
----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 23-Jan-86

Version : 1.00a 27-Jan-86 Paul Curtis, TDI.
Finally gave up on idea of using SetRGB4
for double-buffered displays after about
two hours hacking. SetRGB4 works for non
DB displays; when using DB displays, one
frames colour table is changed, but the
other frames table isn't. Anyway, the new
demo uses two views and two viewports.
They are remade on every frame of the
display. Pretty impressive stuff this!
0.00a 23-Jan-86 Paul Curtis, TDI.
Original

*)

(*\$S-*) (*\$T-*)

IMPORT SYSTEM, GraphicsLibrary, Views, Areas, Colors, Copper,
Rasters, Pens, RandomNumbers, Libraries, MathLib0;

CONST

width = 320;
height = 200;
depth = 2;

MODULE Rotation;


```

FROM MathLib0 IMPORT sin, cos, DegToRad;
FROM RandomNumbers IMPORT Random;
IMPORT x, y, z, nrVertices;

EXPORT YRotation, angle;

CONST
    MaxAngle = 20;
    MaxCount = 8;

VAR
    sinphi, cosphi: ARRAY [0..MaxAngle] OF REAL;
    phi, phidir: INTEGER; (* how much to rotate, and in which direction *)
    angle: INTEGER; (* current facial angle of left side *)
    maxAngle: INTEGER;
    maxCount: CARDINAL;

PROCEDURE YRotation;
    (* rotate cube about y axis by phi degrees. *)

VAR i: CARDINAL;
    X, Z, sphi, cphi: REAL;

BEGIN
    (* get next rotation angle *)
    INC(maxCount);
    IF maxCount >= MaxCount THEN
        maxCount := 0;
        INC(phi, phidir);
        (* NOTE: cannot use ABS(phi) >= maxAngle. *)
        IF (phi <= -maxAngle) & (phidir < 0) OR
            (phi >= maxAngle) & (phidir > 0) THEN
            maxAngle := Random(MaxAngle DIV 2 - 1) + MaxAngle DIV 2 - 1;
            phidir := -phidir;
        END;
    END;

    INC(angle, phi);
    angle := (angle + 90) MOD 90;

    cphi := cosphi[ABS(phi)];
    IF phi < 0 THEN
        sphi := -sinphi[ABS(phi)];
    ELSE

```

```

    sphi := sinphi[phi];
END;
FOR i := 0 TO nrVerticies-1 DO
    X := x[i]; Z := z[i];
    x[i] := X*cphi - Z*sphi;
    z[i] := Z*cphi + X*sphi;
END;
END YRotation;

VAR i: CARDINAL; s: REAL;

BEGIN
    maxAngle := Random(MaxAngle DIV 2-1)+MaxAngle DIV 2 - 1;
    phi := 0; phidir := 1;
    angle := 0;
    maxCount := MaxCount;
    FOR i := 0 TO MaxAngle DO
        sinphi[i] := sin(DegToRad(FLOAT(i)));
        cosphi[i] := cos(DegToRad(FLOAT(i)));
    END;
END Rotation;

MODULE Initialisation;

IMPORT width, height, depth, nrVerticies;
IMPORT GraphicsLibrary, Rasters, Views, Areas, Colors, Libraries, Pens;

FROM SYSTEM IMPORT BYTE, WORD, ADDRESS, ADR, SIZE;
FROM GraphicsLibrary IMPORT GraphicsName, GraphicsBase,
    InitBitMap, BltClear, PlanePtr, BitMap;

EXPORT initOK, RP, VP, V, ColourTable;

VAR
    initOK: BOOLEAN;
    areabuffer: ARRAY [0..99] OF WORD;
    V: ARRAY [0..1] OF Views.View;
    VP: ARRAY [0..1] OF Views.ViewPort;
    CM: ARRAY [0..1] OF Colors.ColorMap;
    RP: ARRAY [0..1] OF Rasters.RastPort;
    RI: ARRAY [0..1] OF Rasters.RasInfo;
    BM: ARRAY [0..1] OF BitMap;
    Depth: ARRAY [0..1] OF CARDINAL;

```

```

ColourTable: ARRAY [0..1] OF ARRAY [0..31] OF CARDINAL;
AI: Areas.AreaInfo;
AP: LONGINT;
TR: Rasters.TmpRas;
TRplane: PlanePtr;

```

```

PROCEDURE InitDemoView(): BOOLEAN;

```

```

VAR i, plane, n: CARDINAL;

```

```

BEGIN

```

```

    GraphicsBase := Libraries.OpenLibrary(GraphicsName,0);

```

```

    IF GraphicsBase = 0 THEN RETURN FALSE; END;

```

```

    FOR i := 0 TO 1 DO

```

```

        InitBitMap(BM[i],depth,width,height);

```

```

        FOR plane := 0 TO depth-1 DO

```

```

            BM[i].Planes[plane] := Rasters.AllocRaster(width,height);

```

```

            IF BM[i].Planes[plane] = 0 THEN

```

```

                Depth[i] := plane; (* nr. planes allocated *)

```

```

                RETURN FALSE;

```

```

            END;

```

```

            (* clear bit plane *)

```

```

            BltClear(BM[i].Planes[plane],

```

```

                LONGCARD(height)*10000H+(LONGCARD(width)+7) DIV 8,{1});

```

```

        END;

```

```

        Depth[i] := depth;

```

```

    END;

```

```

    (* initialise the RasInfo structure *)

```

```

    FOR i := 0 TO 1 DO

```

```

        WITH RI[i] DO

```

```

            bitMap := ADR(BM[i]);

```

```

            rxOffset := 0;

```

```

            ryOffset := 0;

```

```

            next := Rasters.RasInfoPtr(0);

```

```

        END;

```

```

    END;

```

```

    (* initialise the RastPort structures *)

```

```

    FOR i := 0 TO 1 DO

```

```

        Rasters.InitRastPort(RP[i]);

```

```

        RP[i].bitMap := ADR(BM[i]);

```

```

    END;

```

```

(* initialise the view *)
FOR i := 0 TO 1 DO
  Views.InitView(V[i]);
  V[i].viewPort := ADR(VP[i]);

  (* initialise the viewport *)
  Views.InitVPort(VP[i]);
  WITH VP[i] DO
    dWidth := width;
    dHeight := height;
    rasInfo := ADR(RI[i]);
    WITH CM[i] DO
      type := BYTE(0); (* ARRAY xRGB *)
      flags := BYTE(0);
      count := 4;
      colorTable := ADR(ColourTable[i]);
    END;
    colorMap := ADR(CM[i]);
  END;

  Views.MakeVPort(V[i],VP[i]);
  Views.MrgCop(V[i]);
END;

Areas.InitArea(AI,areabuffer,SIZE(areabuffer));
TRplane := Rasters.AllocRaster(width,height);
IF TRplane = 0 THEN RETURN FALSE; END;

Rasters.InitTmpRas(TR,TRplane,(width+15) DIV 16 * height);
AP := -1; (* all bits set *)
FOR i := 0 TO 1 DO
  Pens.SetDrMd(RP[i],GraphicsLibrary.Jan1);
  WITH RP[i] DO
    (* no cross fill in areafilling; not necessary, but makes fill
       run a little faster. *)
    INCL(Flags,Rasters.NoCrossFill);

    (* set area filling info *)
    areaInfo := ADR(AI);
    AreaPtrn := ADR(AP);
    AreaPtSz := BYTE(1);
    tmpRas := ADR(TR);
  END;
END;

```

```

END;

RETURN TRUE;
END InitDemoView;

PROCEDURE FreeMem;
  (* free allocated bitmap space *)
VAR i, plane: INTEGER;
BEGIN
  FOR i := 0 TO 1 DO
    FOR plane := 0 TO Depth[i]-1 DO
      Rasters.FreeRaster(BM[i].Planes[plane],width,height);
    END;
  END;
  Libraries.CloseLibrary(GraphicsBase);
END FreeMem;

BEGIN
  ColourTable[0][3] := 0FH;
  ColourTable[1][3] := 0FH;
  Depth[0] := 0;
  Depth[1] := 0;
  initOK := InitDemoView();
END Initialisation;

MODULE Cube;

FROM Pens IMPORT SetAPen;
FROM Areas IMPORT AreaMove, AreaDraw, AreaEnd;
IMPORT Copper, Views, Rasters;
IMPORT V, VP, RP, depth, width, height, distance, ColourTable,
  YRotation, angle;

EXPORT x, y, z, nrVertices, Do;

CONST
  nrVertices = 4;

VAR
  x, y, z: ARRAY [0..nrVertices-1] OF REAL;
  frame: CARDINAL;
  x2D, y2D: ARRAY [0..nrVertices-1] OF INTEGER;

```

```

PROCEDURE DrawCube;

CONST
  centerX = width DIV 2;
  centerY = height DIV 2;

VAR left: INTEGER;
    i, j, k: CARDINAL;
    vertex: CARDINAL;
    rightObscured: BOOLEAN;
    err: INTEGER;

BEGIN
  frame := 1-frame; (* draw into non-displayed frame *)

  Rasters.SetRast(RP[frame],0);

  (* find leftmost vertex of base plane *)
  left := x2D[0]; i := 0;
  FOR vertex := 1 TO nrVertices-1 DO
    IF x2D[vertex] < left THEN left := x2D[vertex]; i := vertex; END;
  END;

  j := (i+1) MOD nrVertices;
  k := (i+2) MOD nrVertices;

  (* see if right plane is obscured by left plane *)
  rightObscured := x2D[j] >= x2D[k];

  (* draw left visible plane *)
  IF rightObscured THEN SetAPen(RP[frame],3) ELSE SetAPen(RP[frame],1); END;

  err := AreaMove(RP[frame],centerX+x2D[i],centerY+y2D[i]);
  err := AreaDraw(RP[frame],centerX+x2D[j],centerY+y2D[j]);
  err := AreaDraw(RP[frame],centerX+x2D[j],centerY-y2D[j]);
  err := AreaDraw(RP[frame],centerX+x2D[i],centerY-y2D[i]);
  err := AreaDraw(RP[frame],centerX+x2D[i],centerY+y2D[i]);
  AreaEnd(RP[frame]);

  IF NOT rightObscured THEN
    (* draw right visible plane *)
    SetAPen(RP[frame],2);
    err := AreaMove(RP[frame],centerX+x2D[j],centerY+y2D[j]);
    err := AreaDraw(RP[frame],centerX+x2D[k],centerY+y2D[k]);
  
```

```

err := AreaDraw(RP[frame],centerX+x2D[k],centerY-y2D[k]);
err := AreaDraw(RP[frame],centerX+x2D[j],centerY-y2D[j]);
AreaEnd(RP[frame]);
END;

(* free all old view copper lists. *)
Views.FreeVPortCopLists(VP[frame]);
Copper.FreeCprList(V[frame].LOFCprList^);
Copper.FreeCprList(V[frame].SHFCprList^);

(* calculate new colour values *)
ColourTable[frame][2] := angle DIV 8 + 4;
ColourTable[frame][1] := (89-angle) DIV 8 + 4;

(* make a new viewport *)
V[frame].LOFCprList := Copper.cprlistptr(0);
V[frame].SHFCprList := Copper.cprlistptr(0);

Views.MakeVPort(V[frame],VP[frame]);
Views.MrgCop(V[frame]);

(* load the new frame on flyback *)
Views.LoadView(V[frame]);

Views.WaitTOF;

END DrawCube;

PROCEDURE Convert2D;
(* project 3D image onto 2D surface. *)

VAR i: CARDINAL; f: REAL;

PROCEDURE Transform(x, f: REAL): INTEGER;
(* this procedure is used as TRUNC expects a positive REAL
and returns a CARDINAL. This is not an implementation
restriction: it is defined in the "Report on The
Programming Language Modula-2" by N. Wirth. *)

VAR t: REAL;

BEGIN
t := x*f;
IF t < 0.0 THEN

```

```

        RETURN -INTEGER(TRUNC(ABS(t)));
    ELSE
        RETURN INTEGER(TRUNC(t));
    END;
END Transform;

BEGIN
    FOR i := 0 TO nrVertices-1 DO
        f := 1000.0 / (distance - z[i]);
        x2D[i] := Transform(x[i],f);
        y2D[i] := Transform(y[i],f);
    END;
END Convert2D;

PROCEDURE Do(n: CARDINAL);
VAR i: CARDINAL;
BEGIN
    WHILE n > 0 DO
        YRotation; Convert2D; DrawCube;
        DEC(n);
    END;
END Do;

BEGIN
    frame := 0;
    x[0] := -150.0; y[0] := 150.0; z[0] := 150.0;
    x[1] := 150.0; y[1] := 150.0; z[1] := 150.0;
    x[2] := 150.0; y[2] := 150.0; z[2] := -150.0;
    x[3] := -150.0; y[3] := 150.0; z[3] := -150.0;
END Cube;

VAR c: CARDINAL;
    distance, dir: REAL;

PROCEDURE NewDir(): REAL;
BEGIN
    RETURN FLOAT(RandomNumbers.Random(60)+40);
END NewDir;

BEGIN
    IF initOK THEN

```


(* just rotate on spot at first *)

distance := 6500.0; Do(40);

dir := NewDir();

LOOP

Do(1);

distance := distance + dir;

IF distance <= 1800.0 THEN

dir := NewDir(); Do(RandomNumbers.Random(30)+20);

ELSIF distance >= 6500.0 THEN

dir := -NewDir(); Do(RandomNumbers.Random(30)+20);

END;

END;

END;

END Cube.

This page intentionally left blank

Appendix B

Modula-2/Amiga Run-time Licence

Programs developed using TDI Modula-2/Amiga are stand-alone native code programs that do not require the TDI Modula-2/Amiga system to run. You are free to develop and market products written in TDI Modula-2/Amiga without any run-time licence charge. We do ask that you include the following reference in the manual or information box of the product.

"Developed using TDI Modula-2/Amiga.
TDI Software Inc. Dallas, USA."

This page intentionally left blank

Appendix C

Modula-2 Bibliography

Prof. N. Wirth,
Programming in Modula-2
(Springer Verlag, 1985 0-387-12206-0)

This is a standard reference to Modula-2. It defines the language in a clear and unambiguous way, and is an invaluable reference for anyone who is writing serious programs in Modula-2. It is, however, a little terse and so not a good book from which to learn how to write computer programs.

Ogilvie,
Modula-2 Programming
(McGraw Hill 0-07-047770-1)

A very complete hardback book. Covers the entire language from simple usage to complex systems design.

R. Gleaves,
Modula-2 for Pascal Programmers
(Springer Verlag, 1984)

This is a useful book for someone who has learned Pascal, and wants to start using Modula-2.

Knepley and Platt,
Modula-2 Programming
(Prentice Hall 0-8359-4602-9)

Thalmann,
Modula-2 an Introduction
(Springer Verlag 0-387-13297-X)

This page intentionally left blank

Appendix D

This appendix contains the source of the definition modules for TDI Modula-2/Amiga. Both the standard library and the Amiga interface modules are included. The end of the appendix contains a cross reference listing of all the items exported from the modules.

DEFINITION MODULE ADKBits;

(** -----
Commodore Amiga adkcon hardware register module
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved
----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 29-Jan-86

Version : 0.00a 29-Jan-86 Paul Curtis, TDI.
Original

*)

TYPE

ADKBit = (Use0V1, (* use audio chan 0 to modulate volume of 1 *)
Use1V2, (* use audio chan 1 to modulate volume of 2 *)
Use2V3, (* use audio chan 2 to modulate volume of 3 *)
Use3Vn, (* use audio chan 3 to modulate volume of ? *)
Use0P1, (* use audio chan 0 to modulate period of 1 *)
Use1P2, (* use audio chan 1 to modulate period of 2 *)
Use2P3, (* use audio chan 2 to modulate period of 3 *)
Use3Pn, (* use audio chan 3 to modulate period of ? *)
Fast, (* 1 -> 2 us/bit (mfm), 2 -> 4 us/bit (gcr) *)
MSBSync, (* (Apple GCR Only) sync on MSB for reading *)
WordSync, (* enable DSKSYNC register matching *)
UartBrk, (* force uart output to zero *)
MFMPrec, (* use mfm style precompensation *)
Precomp0,
Precomp1, (* two bits of precompensation *)
ADKSetClr); (* standard set/clear bit *)

TYPE

ADKBitSet = SET OF ADKBit;

CONST

(* precompensation timings. *)


```
Pre000NS = ADKBitSet{};           (* 000 ns of precomp *)
Pre140NS = ADKBitSet{Precomp0};    (* 140 ns of precomp *)
Pre280NS = ADKBitSet{Precomp1};    (* 280 ns of precomp *)
Pre560NS = ADKBitSet{Precomp0,Precomp1}; (* 560 ns of precomp *)
```

```
END ADKBits.
```

DEFINITION MODULE Alerts;

(** -----

Commodore Amiga alerts module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 30-Dec-85

Version : 0.00a 30-Dec-85 Paul Curtis, TDI.
Original

*)

CONST

(* alert types *)

ATDeadEnd = 80000000H;

ATRecovery = 0H;

CONST

(* general alert codes *)

AGNoMemory = 10000H;

AGMakeLib = 20000H;

AGOpenLib = 30000H;

AGOpenDev = 40000H;

AGOpenRes = 50000H;

AGIOError = 60000H;

CONST

(* alert origin within system, i.e. where alert was generated *)

AOExecLib = 8001H;

AOGraphicsLib = 8002H;

AOLayersLib = 8003H;

AOIntuition = 8004H;

AOMathLib = 8005H;

AOCLib = 8006H;

AODOSLib = 8007H;

```

ADRAMLib      = 8008H;
ADIconLib     = 8009H;
ADAudioDev    = 8010H;
ADConsoleDev  = 8011H;
ADGamePortDev = 8012H;
ADKeyboardDev = 8013H;
ADTrackDiskDev = 8014H;
ADTimerDev    = 8015H;
ADCIARsrc     = 8020H;
ADDiskRsrc    = 8021H;
ADMiscRsrc    = 8022H;
ADBootStrap   = 8030H;
ADWorkbench   = 8031H;

```

CONST

(* special alerts for the exec library *)

```

ANExecLib     = 01000000H;
ANExcptVect   = 81000001H; (* 68000 exception vector checksum *)
ANBaseChkSum  = 81000002H; (* exexbase checksum *)
ANLibChkSum   = 81000003H; (* library checksum failure *)
ANLibMem      = 81000004H; (* no memory to make library *)
ANMemCorrupt  = 81000005H; (* corrupted memory list *)
ANIntrMem     = 81000006H; (* no memory for interrupt servers *)

```

CONST

(* special alerts for the graphics library *)

```

ANGraphicsLib = 02000000H;
ANCopDisplay  = 82010001H; (* copper display list, no memory *)
ANCopInstr    = 82010002H; (* copper instruction list, no memory *)
ANCopListOver = 82000003H; (* copper list overload *)
ANCopIListOver = 82000004H; (* copper intermediate list overload *)
ANCopListHead = 82010005H; (* copper list head, no memory *)
ANLongFrame   = 82010006H; (* long frame, no memory *)
ANShortFrame  = 82010007H; (* short frame, no memory *)
ANFloodFill   = 82010008H; (* flood fill, no memory *)
ANTextTmpRas  = 02010009H; (* text, no memory for TmpRas *)
ANBltBitMap   = 8201000AH; (* BltBitMap, no memory *)

```

CONST

(* special alerts for the layer library *)

```

ANLayersLib   = 03000000H;

```

CONST

(* special alerts for the intuition library *)

```
ANIntuition      = 04000000H;
ANGadgetType     = 04000001H; (* unknown gadget type *)
ANBadGadget      = 04000001H; (* recovery form of ANGadgetType *)
ANCreatePort     = 04010002H; (* create port, no memory *)
ANItemAlloc      = 04010003H; (* item plane alloc, no memory *)
ANSubAlloc       = 04010004H; (* sub alloc, no memory *)
ANPlaneAlloc     = 04010005H; (* plane alloc, no memory *)
ANItemBoxTop     = 04000006H; (* item box top < RelZero *)
ANOpenScreen     = 04010007H; (* open screen, no memory *)
ANOpenScrnRast   = 04010008H; (* open screen, raster alloc, no memory *)
ANSysScrnType    = 04000009H; (* open sys screen, unknown type *)
ANAddSWGadget    = 0401000AH; (* add SW gadgets, no memory *)
ANOpenWindow     = 0401000BH; (* open window, no memory *)
ANBadState       = 0400000CH; (* bad state return entering Intuition *)
ANBadMessage     = 0400000DH; (* bad message received by IDCMP *)
ANWeirdEcho      = 0400000EH; (* weird echo causing incomprehension *)
ANNoConsole      = 0400000FH; (* couldn't open the console device *)
```

CONST

(* special alerts for the maths library *)

```
ANMathLib        = 05000000H;
```

CONST

(* special alerts for the character list library *)

```
ANCListLib       = 06000000H;
```

CONST

(* special alerts for the DOS library *)

```
ANDOSLib         = 07000000H;
ANStartMem       = 07010001H; (* no memory at startup *)
ANEndTask        = 07000002H; (* EndTask didn't *)
ANOPktFail       = 07000003H; (* opkt failure *)
ANAsyncPkt       = 07000004H; (* unexpected packet received *)
ANFreeVec        = 07000005H; (* freevec failed *)
ANDiskBlkSeq     = 07000006H; (* disk block sequence error *)
ANBitMap         = 07000007H; (* bitmap corrupt *)
ANKeyFree        = 07000008H; (* key already free *)
ANBadChkSum      = 07000009H; (* Invalid checksum *)
ANDiskError      = 0700000AH; (* disk error *)
ANKeyRange       = 0700000BH; (* key out of range *)
```

```
ANBadOverlay = 07000000H; (* bad overlay *)
```

```
CONST
```

```
(* special alerts for the RAM library *)
```

```
ANRAMLib = 08000000H;
```

```
CONST
```

```
(* special alerts for the icon library *)
```

```
ANIconLib = 09000000H;
```

```
CONST
```

```
(* special alerts for the audio device *)
```

```
ANAudioDev = 10000000H;
```

```
CONST
```

```
(* special alerts for the console device *)
```

```
ANConsoleDev = 11000000H;
```

```
CONST
```

```
(* special alerts for the gameport device *)
```

```
ANGamePortDev = 12000000H;
```

```
CONST
```

```
(* special alerts for the keyboard device *)
```

```
ANKeyboardDev = 13000000H;
```

```
CONST
```

```
(* special alerts for the trackdisk device *)
```

```
ANTrackDiskDev = 14000000H;
```

```
ANTDCalibSeek = 14000001H; (* calibrate: seek error *)
```

```
ANTDDelay = 14000002H; (* delay: error on timer wait *)
```

```
CONST
```

```
(* special alerts for the timer device *)
```

```
ANTimerDev = 15000000H;
```

```
ANtMBadReq = 15000001H; (* bad request *)
```

```
CONST
```

```
(* special alerts for the cia resource *)
```

```
ANCIARsrc = 20000000H;
```

```
CONST
```

```
(* special alerts for the disk.r source *)
```

```
ANDiskRsrc = 21000000H;
```

```
ANDRHasDisk    = 21000001H;  (* get unit: already has disk *)
ANDRIntNoAct   = 21000002H;  (* interrupt: no active unit *)
```

CONST

```
(* special alerts for the misc.r source *)
ANMiscRsrc     = 22000000H;
```

CONST

```
(* special alerts for the bootstrap *)
ANBootStrap    = 30000000H;
ANBootError    = 30000001H;  (* boot code returned an error *)
```

CONST

```
(* special alerts for the Workbench *)
ANWorkbench    = 31000000H;
```

END Alerts.

DEFINITION MODULE AmigaUtils;

(** -----

Commodore Amiga utilities module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 03-Jan-86

Version : 0.00a 03-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS, LONGWORD;
FROM DOSLibrary IMPORT BPTR;

PROCEDURE BPTRFromPtr(ptr: ADDRESS): BPTR;

(* convert a BCPL pointer into a Modula or C pointer.

ptr: the Modula or C pointer to convert.

returns: the BCPL pointer. *)

PROCEDURE PtrFromBPTR(BPtr: BPTR): ADDRESS;

(* convert a Modula or C pointer into a BCPL pointer.

Bptr: the BCPL pointer to convert.

returns: the Modula or C pointer. *)

PROCEDURE NULL(x: LONGWORD): BOOLEAN;

(* see if a long word is a C NULL pointer.

x: the long word to test.

returns: TRUE => x was NULL, otherwise FALSE. *)

END AmigaUtils.

DEFINITION MODULE AMIGAX;

(** -----

Commodore Amiga run time support module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : ETH, Zurich.

Modifications : GEM Adaption, Phil Camp, TDI Software, Inc.
Amiga Adaption, Paul Curtis, TDI Software, Inc.

Version : 4.00b 09-Jan-86 Paul Curtis TDI.
Added ExitM2 PROC variable as stacked return
address was corrupted on an error.
4.00a 04-Dec-85 Paul Curtis, TDI.
Original Amiga implementation.

*)

(* NB. The sequence of the definitions in this module are fixed.
It is essential that they reflect the declarations in
the Modula compiler.

*)

FROM SYSTEM IMPORT LONGWORD, ADDRESS, PROCESS;

TYPE

ErrorCause = (XError, XRaise, XPropagate);
ErrorType = (EM1111TRAP, EM1010TRAP, PrivilegeViolationTRAP,
ArithOverflowTRAP, OutOfRangeTRAP, ZeroDivideTRAP,
IllegalInstrTRAP, AdrErrorTRAP, BusErrorTRAP,
ProgramHalt, NoFunctionReturn, CaseIndexRange,
StackOverflow, OutOfRange, ArithOverflow,
NewprocessWorkspace, ProcessTerminated, UnimplementedRoutine,
NormalReturn, User1, User2, User3, User4);

ErrorContextType = RECORD

Error : INTEGER; (* ErrorType... *)
CameFrom : ErrorCause;

```

PC      : ADDRESS;
SR      : CARDINAL;
A5, A6, A7: ADDRESS;
LastMP  : ADDRESS;
DumpToDo : BOOLEAN;
FileToDump: ARRAY [0..22] OF CHAR;
END;

```

```
ErrorProcessorType = PROCEDURE () ;
```

```
CONST
```

```
  AbsExecBase = 4;
```

```
VAR
```

```

ExecBase[AbsExecBase]: ADDRESS; (* exec library base *)
CLinePtr: ADDRESS;      (* pointer to command line arguments *)
CLineLen: LONGCARD;     (* length of command line *)
StackSize: LONGCARD;    (* size of stack in bytes *)
StackPtr: ADDRESS;      (* user stack pointer on entry to task *)

```

```
(* user supplied run-time error processor *)
```

```
ErrorProcessor: ErrorProcessorType;
```

```
ErrorContext: ErrorContextType; (* Error details *)
```

```

ExitM2: PROC; (* If this procedure is called, the program will stop
               running and return. The return code should be put
               into D0. This is not normally necessary unless an
               error has been detected. For example:

```

```

    normal exit:
        SETREG(0,0);
        AMIGAX.ExitM2;

```

```

    abnormal exit, send back returncode -7:
        SETREG(0,-7);
        AMIGAX.ExitM2; *)

```

```
(* UTILITIES SUPPORTING THE COMPILER *)
```

```

PROCEDURE CASEX;
PROCEDURE HALTX;
PROCEDURE STACKTEST;

```

(* PSEUDO MODULE SYSTEM *)

```
PROCEDURE NEWPROCESS (PROCESSCODE : PROC;  
                     FPAWSP : ADDRESS; LENGTHWSP : LONGCARD;  
                     VAR PROCESSDESCRIPTOR : PROCESS;  
                     INITIALPRIO : LONGCARD);
```

```
PROCEDURE TRANSFER;  
PROCEDURE IOTRANSFER;  
PROCEDURE SYSCALL;
```

(* 32-BIT (LONG-) INTEGER/CARDINAL ARITHMETIC *)

```
PROCEDURE MULU32      (MULTIPlicAN, MULTIPLIER : LONGCARD);  
PROCEDURE DIVU32      (DIVIDEND, DIVISOR      : LONGCARD);  
PROCEDURE MULS32      (MULTIPlicAN, MULTIPLIER : LONGINT);  
PROCEDURE DIVS32      (DIVIDEND, DIVISOR      : LONGINT);
```

(* 32-BIT IEEE FLOATING POINT FORMAT ARITHMETIC *)

```
PROCEDURE FADD        (ADDER,      ADDEND      : REAL);  
PROCEDURE FSUB        (MINUEND,    SUBTRAHEND  : REAL);  
PROCEDURE FMUL        (MULTIPlicAN, MULTIPLIER : REAL);  
PROCEDURE FDIV        (DIVIDEND,    DIVISOR    : REAL);  
PROCEDURE FCMP        (DESTINATION, SOURCE     : REAL);  
PROCEDURE FTST        (TOTEST      : REAL);  
PROCEDURE FLOATX      (TOCONVERT    : LONGWORD);  
PROCEDURE TRUNCX      (TOCONVERT    : REAL);
```

(* 64-BIT IEEE FLOATING POINT FORMAT ARITHMETIC *)

(* These routines are not supported by the compiler. They generate errors if called from user programs by direct importation -- PLC *)

```
PROCEDURE LFADD        (ADDER,      ADDEND      : LONGREAL);  
PROCEDURE LFSUB        (MINUEND,    SUBTRAHEND  : LONGREAL);  
PROCEDURE LFMUL        (MULTIPlicAN, MULTIPLIER : LONGREAL);  
PROCEDURE LFDIV        (DIVIDEND,    DIVISOR    : LONGREAL);  
PROCEDURE LFCMP        (DESTINATION, SOURCE     : LONGREAL);  
PROCEDURE LFTST        (TOTEST      : LONGREAL);  
PROCEDURE LFLOATX      (TOCONVERT    : LONGWORD);  
PROCEDURE LTRUNCX      (TOCONVERT    : LONGREAL);
```

END (* OF DEFINITION MODULE *) AMIGAX.

DEFINITION MODULE ANSIConsole;

```
(** -----  
Commodore Amiga ANSI console module  
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved  
----- **)
```

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 29-Jan-86

Version : 0.00a 29-Jan-86 Paul Curtis, TDI.
Original

*)

(* These commands are caught by the console driver and
perform the relevent actions. *)

CONST

(* independent control functions, no introducer *)

BS = 10C; (* backspace *)

LF = 12C; (* line feed *)

VT = 13C; (* vertical tab *)

FF = 14C; (* form feed *)

CR = 15C; (* carriage return *)

SO = 16C; (* shift out *)

SI = 17C; (* shift in *)

ESC = 33C; (* escape *)

CONST

IND = 204C; (* index: move the active position down one line, also ESC D *)

NEL = 205C; (* next line, also ESC E *)

RI = 215C; (* reverse index *)

CSI = 233C; (* control sequence introducer *)

CONST

(* ISO compatible escape sequences, introduced by ESC *)

INT = "a"; (* interrupt *)

RIS = "c"; (* reset to initial state *)

CONST

(* control sequence, introduced by CSI.

1: optional parameter, e.g. insert one char = CSI ESC @

insert two chars = CSI ESC 2 @

2: two parameters, e.g. cursor position at 10,20 = CSI CUP 10;20H

>2: many parameters, e.g. set graphic rendition. *)

ICH = "@"; (* 1: insert character *)

CUU = "A"; (* 1: cursor up *)

CUD = "B"; (* 1: cursor down *)

CUF = "C"; (* 1: cursor forward *)

CUB = "D"; (* 1: cursor backward *)

CNL = "E"; (* 1: cursor next line *)

CPL = "F"; (* 1: cursor preceeding line *)

CUP = "H"; (* 2: cursor position *)

ED = "J"; (* 1: erase in display (to end of display) *)

EL = "K"; (* 1: erase in line (to end of line) *)

IL = "L"; (* 1: insert line *)

DL = "M"; (* 1: delete line *)

DCH = "P"; (* 1: delete character *)

CPR = "R"; (* 2: cursor position report, read stream *)

SU = "S"; (* 1: scroll up *)

SD = "T"; (* 1: scroll down *)

SM = "h"; (* 3: set mode *)

RM = "l"; (* 3: reset mode *)

CONST

(* private Amiga control sequence, introduced by CSI. *)

aSLPP = "t"; (* 1: set lines per page *)

aSLL = "u"; (* 1: set line length *)

aSLO = "x"; (* 1: set line offset *)

aSTO = "y"; (* 1: set top offset *)

aSRE = "{"; (* 3: set raw events *)

aIER = "|"; (* 8: input event report, read sequence *)

aRRE = "}"; (* 3: reset raw events *)

aSKR = "~"; (* 1: special key report, read sequence *)

aSCR = "p"; (* 1: set cursor rendition *)

aWSR = "q"; (* 0: window status request *)

aWBR = "r"; (* window bounds report, read stream *)

END ANSIConsole.

DEFINITION MODULE ANSIPrinter;

(** -----

Commodore Amiga ANSI printer module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 29-Jan-86

Version : 0.00a 29-Jan-86 Paul Curtis, TDI.
Original

*)

(* These commands are caught by the printer driver and
perform the relevent actions. *)

CONST

RIS = "c"; (* reset *)
RIN = "#1"; (* initialize *)
IND = "D"; (* line feed *)
NEL = "E"; (* return, line feed *)
RI = "M"; (* reverse line feed *)

SGR0 = "[0m"; (* normal character set *)
SGR3 = "[3m"; (* italics on *)
SGR23 = "[23m"; (* italics off *)
SGR4 = "[4m"; (* underline on *)
SGR24 = "[24m"; (* underline off *)
SGR1 = "[1m"; (* boldface on *)
SGR22 = "[22m"; (* boldface off *)
SFC = "30-39"; (* set foreground color *)
SBC = "40-49"; (* set background color *)

SHORP0 = "[0w"; (* normal pitch *)
SHORP2 = "[2w"; (* elite on *)
SHORP1 = "[1w"; (* elite off *)
SHORP4 = "[4w"; (* condensed fine on *)

SHORP3 = "[3w"; (* condensed off *)
 SHORP6 = "[6w"; (* enlarged on *)
 SHORP5 = "[5w"; (* enlarged off *)

 DEN6 = "[6"z"; (* shadow print on *)
 DEN5 = "[5"z"; (* shadow print off *)
 DEN4 = "[4"z"; (* doublestrike on *)
 DEN3 = "[3"z"; (* doublestrike off *)
 DEN2 = "[2"z"; (* near letter quality on *)
 DEN1 = "[1"z"; (* near letter quality off *)

 SUS2 = "[2v"; (* superscript on *)
 SUS1 = "[1v"; (* superscript off *)
 SUS4 = "[4v"; (* subscript on *)
 SUS3 = "[3v"; (* subscript off *)
 SUS0 = "[0v"; (* normalize the line *)
 PLU = "L"; (* partial line up *)
 PLD = "K"; (* partial line down *)

 FNT0 = "(B"; (* US char set *)
 FNT1 = "(R"; (* French char set *)
 FNT2 = "(K"; (* German char set *)
 FNT3 = "(A"; (* UK char set *)
 FNT4 = "(E"; (* Danish I char *)
 FNT5 = "(H"; (* Sweden char *)
 FNT6 = "(Y"; (* Italian char set *)
 FNT7 = "(Z"; (* Spanish char set *)
 FNT8 = "(J"; (* Japanese char set *)
 FNT9 = "(6"; (* Norweign char set *)
 FNT10 = "(C"; (* Danish II char set *)

 PROP2 = "[2p"; (* proportional on *)
 PROP1 = "[1p"; (* proportional off *)
 PROPB = "[0p"; (* proportional clear *)
 (*TSS = "[n E"; (* set proportional offset *) *)
 JFY5 = "[5 F"; (* auto left justify *)
 JFY7 = "[7 F"; (* auto right justify *)
 JFY6 = "[6 F"; (* auto full justify *)
 JFY0 = "[0 F"; (* auto justify off *)
 JFY3 = "[3 F"; (* letter space, justify *)
 JFY1 = "[1 F"; (* word fill, auto center *)

 VERPB = "[0z"; (* 1/8" line spacing *)


```

VERP1 = "[1z";      (* 1/6" line spacing *)
(*SLPP = "[nt";      (* set form length n *) *)
(*PERF = "[nq";      (* perf skip n (n>0) *) *)
PERF0 = "[0q";      (* perf skip off *)

LMS = "#9";         (* left margin set *)
RMS = "#0";         (* right margin set *)
TMS = "#8";         (* top margin set *)
BMS = "#2";         (* bottom marg set *)
(*STBM = "[Pn1;Pn2r";(* top and bottom margins *) *)
(*SLRM = "[Pn1;Pn2s";(* left and right margins *) *)
CAM = "#3";         (* clear margins *)

HTS = "H";          (* set horiz tab *)
VTS = "J";          (* set vertical tabs *)
TBC0 = "[0g";       (* clear horizontal tab *)
TBC3 = "[3g";       (* clear all horizontal tabs *)
TBC1 = "[1g";       (* clear vertical tabs *)
TBC4 = "[4g";       (* clear all vertical tabs *)
TBCALL = "#4";      (* clear all horizontal & vertical tab *)
TBSALL = "#5";      (* set default tabs *)

```

```

END ANSIPrinter.

```

DEFINITION MODULE Areas;

```
(** -----  
Commodore Amiga area fill module  
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved  
----- **)
```

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 16-Jan-86

Version : 0.00a 16-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT WORD, ADDRESS;
FROM Rasters IMPORT RastPort;

TYPE

AreaInfoPtr = POINTER TO AreaInfo;

AreaInfo = RECORD

 vctrTbl: ADDRESS; (* pointer to start of vector table *)
 vctrPtr: ADDRESS; (* pointer to current vertex *)
 flagTbl: ADDRESS; (* pointer to start of vector flag table *)
 flagPtr: ADDRESS; (* pointer to areafill flags *)
 count: CARDINAL; (* nr. vertices in list *)
 MaxCount: CARDINAL; (* Count <= MaxCount *)
 FirstX: CARDINAL; (* first point for this polygon *)
 FirstY: CARDINAL;
END;

PROCEDURE AreaDraw(VAR rp: RastPort; x,y: CARDINAL): INTEGER;

(* add a point to the list of end points for area fill.

rp: the rastport to draw in.

x,y: the end point coordinate pair.

returns: 0 => point added OK, otherwise vector table too small. *)

```

PROCEDURE AreaMove(VAR rp: RastPort; x,y: CARDINAL): INTEGER;
  (* define a new starting point for a new shape in the vector list.

  rp: the rastport to draw in.
  x,y: the initial point coordinate pair.

  returns: 0 => point added OK, otherwise vector table too small. *)

```

```

PROCEDURE AreaEnd(VAR rp: RastPort);
  (* process table of vectors and produce areafill.

```

```

    rp: the rastport to process. *)

```

```

PROCEDURE InitArea(VAR ai: AreaInfo; VAR buf: ARRAY OF WORD; size: LONGCARD);
  (* initialise vector collection matrix.

```

```

    ai: the AreaInfo structure to initialise.
    buf: the collection place for vectors.
    size: the size of the buffer in bytes. *)

```

```

END Areas.

```

DEFINITION MODULE ASCII;

(* Defines standard names for ASCII codes *)

(**-----**)

(* (c) Copyright 1984 32DOS Ltd All Rights Reserved *)

(**-----**)

(* (c) Copyright 1985 TDI Ltd All Rights Reserved *)

(**-----**)

EXPORT QUALIFIED

(* CONST *) NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL,
BS, HT, LF, VT, FF, CR, SO, SI,
DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
CAN, EM, SUB, ESC, FS, GS, RS, US, DEL,
OctalChars, DecimalChars, HexadecimalChars,
AlphabetChars, RealChars,

(* TYPE *) CharCases,

(* PROC *) CharIsPrintable, CharIsControl, CharIsASCII;

CONST NUL = 00C; SOH = 01C; STX = 02C; ETX = 03C;
EOT = 04C; ENQ = 05C; ACK = 06C; BEL = 07C;
BS = 08C; HT = 09C; LF = 0AC; VT = 0BC;
FF = 0CC; CR = 0DC; SO = 0EC; SI = 0FC;
DLE = 10C; DC1 = 11C; DC2 = 12C; DC3 = 13C;
DC4 = 14C; NAK = 15C; SYN = 16C; ETB = 17C;
CAN = 18C; EM = 19C; SUB = 1AC; ESC = 1BC;
FS = 1CC; GS = 1DC; RS = 1EC; US = 1FC;
DEL = 177C;

OctalChars = '01234567';
DecimalChars = '0123456789';
HexadecimalChars = '0123456789ABCDEF';
AlphabetChars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
RealChars = '0123456789.E+-';

TYPE CharCases = (Upper, Lower, UpperOrLower, Shift, Control, ControlShift);

PROCEDURE CharIsPrintable((* checks *) Ch : CHAR) : BOOLEAN;

```
(* Returns TRUE if Ch is printable and ASCII *)
```

```
PROCEDURE CharIsControl((* checks *) Ch : CHAR) : BOOLEAN;
```

```
(* Returns TRUE if Ch is ASCII control char *)
```

```
PROCEDURE CharIsASCII((* checks *) Ch : CHAR) : BOOLEAN;
```

```
(* Returns TRUE if (CharIsPrintable OR CharIsControl) *)
```

```
END (* OF MODULE *) ASCII.
```

DEFINITION MODULE AudioDevice;

```
(** -----  
Commodore Amiga audio device module  
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved  
----- **)
```

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 16-Dec-85

Version : 0.00a 16-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;
FROM IO IMPORT CmdNonStd, IORequest;
FROM Ports IMPORT Message;

CONST

AudioName = "audio.device";

(* number of hardware channels *)

ADHardChannels = 4;

ADAllocMinPrec = -128;

ADAllocMaxPrec = 127;

(* Audio device commands *)

ADCmdFree = CmdNonStd + 0;

ADSetPrec = CmdNonStd + 1;

ADCmdFinish = CmdNonStd + 2;

ADCmdPerVol = CmdNonStd + 3;

ADCmdLock = CmdNonStd + 4;

ADCmdWaitCycle = CmdNonStd + 5;

CONST

ADCmdNoUnit = 5;

ADCmdAllocate = 5;

CONST

ADIOPerVol = 4;
ADIOSyncCycle = 5;
ADIOBNoWait = 6;
ADIOBWriteMessage = 7;

CONST

ADIOErrNoAllocation = -10;
ADIOErrAllocFailed = -11;
ADIOErrChannelStolen = -12;

TYPE

AudioChannels = (Left0, Left1, Right0, Right1);
AudioChannelSet = SET OF AudioChannels;

TYPE

IOAudio = RECORD
 ioaRequest: IORequest;
 ioaAllocKey: CARDINAL;
 ioaData: ADDRESS;
 ioaLength: LONGCARD;
 ioaPeriod: CARDINAL;
 ioaVolume: CARDINAL;
 ioaCycles: CARDINAL;
 ioaWriteMsg: Message;
END;

END AudioDevice.

DEFINITION MODULE Blitter;

```
(** -----  
Commodore Amiga graphics blitter module  
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved  
----- **)
```

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 21-Jan-86

Version : 0.00a 21-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, WORD, ADDRESS;
FROM Rasters IMPORT RastPort;

CONST
CLEANUP = 040H;

TYPE
bltnodeptr = POINTER TO bltnode;
bltnode = RECORD
n: bltnodeptr;
function: PROCEDURE; (* DO contains return value *)
stat: BYTE; (* CLEANUP => call cleanup procedure *)
blitsize: INTEGER;
beamsync: INTEGER;
cleanup: PROCEDURE;
END;

PROCEDURE BltPattern(VAR rp: RastPort; VAR mask: ARRAY OF WORD;
x1,y1: CARDINAL; maxx, maxy: CARDINAL;
bytecnt: LONGCARD);
(* using rules for areafill, blit through a mask.

rp: The rastport to blit into.
mask: the 2D mask to blit using.
x1,y1: the upper left corner of the blit area.
maxx,maxy: the lower right corner of the blit area.
bytecnt: nr. bytes per row in mask. *)

PROCEDURE DisownBlitter;
(* return blitter to free state. *)

PROCEDURE OwnBlitter;
(* get blitter for private usage. *)

PROCEDURE QBlit(VAR bp: bltnode);
(* queue up a request for blitter usage.
bp: the initialised blitter request node. *)

PROCEDURE QBSBlit(VAR bp: bltnode);
(* synchronise the blitter request with the video beam.
bp: the initialised blitter request node. *)

PROCEDURE VBeamPos(): CARDINAL;
(* get vertical beam position at this instant. *)

PROCEDURE WaitBlit;
(* wait for the blitter to be finished. *)

END Blitter.

DEFINITION MODULE BlitterHardware;

```
(** -----  
Commodore Amiga blitter hardware module  
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved  
----- **)
```

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 29-Jan-86
Version : 0.00a 29-Jan-86 Paul Curtis, TDI.
Original

*)

CONST

HSizeBits = 6;
VSizeBits = 16-HSizeBits;
HSizeMask = 03FH;
VSizeMask = 03FFH;

CONST

MaxBytesPerRow = 128;

TYPE

(* blitter control register 0 *)
BplCon0 = (NANBNC, NANBC, NABNC, NABC, ANBNC, ANBC, ABNC, ABC,
BC0B0Dest, BC0BSrcC, BC0BSrcB, BC0BSrcA);
BplCon0Set = SET OF BplCon0;

CONST

AorB = BplCon0Set{ABC,ANBC,NABC , ABNC,ANBNC,NABNC};
AorC = BplCon0Set{ABC,NABC,ABNC , ANBC,NANBC,ANBNC};
AtoD = BplCon0Set{ABC,ANBC,ABNC,ANBNC};
AxorC = BplCon0Set{NABC,ABNC , NANBC,ANBNC};

CONST

Dest = 100H;
SrcC = 200H;

```
SrcB = 400H;  
SrcA = 800H;
```

```
CONST
```

```
AShiftShift = 12; (* bits to right align ashift value *)  
BShiftShift = 12; (* bits to right align bshift value *)
```

```
TYPE
```

```
(* blitter control register 0 *)
```

```
BplCon1 = (LineMode, OneDot, FillCarryIn, FillOR, FillXOR, OvFlag, SignFlag);  
BplCon1Set = SET OF BplCon1;
```

```
CONST
```

```
SUD = 10H;  
SUL = 08H;  
AUL = 04H;
```

```
CONST
```

```
Octant8 = 18H;  
Octant7 = 04H;  
Octant6 = 0CH;  
Octant5 = 1CH;  
Octant4 = 14H;  
Octant3 = 08H;  
Octant2 = 00H;  
Octant1 = 10H;
```

```
END BlitterHardware.
```

DEFINITION MODULE CIAHardware;

(** -----

Commodore Amiga CIA hardware module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 30-Dec-85

Version : 0.00a 30-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE;

TYPE

(* general types *)

Pad = ARRAY [1..255] OF BYTE;

SHORTSET = SET OF [0..7];

TYPE

(* CIA interrupt control bits *)

CIAICRBits = (TA, TB, ALRM, SP, FLG, IR, SETCLR);

CIAICR = SET OF CIAICRBits;

TYPE

(* control registers A and B *)

CIACRBits = (Start, PBOOn, OutMode, RunMode, Load, InMd0, InMd1, TODAlarm);

CIACR = SET OF CIACRBits;

TYPE

CIAType = RECORD

ciapra: SHORTSET; pad0: Pad;

ciaprb: SHORTSET; pad1: Pad;

ciaddr: BYTE; pad2: Pad;

ciaddrb: BYTE; pad3: Pad;

```

        ciatalo: BYTE;    pad4: Pad;
        ciatahi: BYTE;    pad5: Pad;
        ciatblo: BYTE;    pad6: Pad;
        ciatbhi: BYTE;    pad7: Pad;
        ciatodlow: BYTE;  pad8: Pad;
        ciatodmid: BYTE;  pad9: Pad;
        ciatodhi: BYTE;   padA: Pad;
        unusedreg: BYTE;  padB: Pad;
        ciasdr: BYTE;     padC: Pad;
        ciaicr: CIACR;    padD: Pad;
        ciacra: CIACR;    padE: Pad;
        ciacrb: CIACR;

END;

```

CONST

```

(* CRB modes *)
CIACRBInPhi2 = CIACR{-};
CIACRBInCnt  = CIACR{InMd0};
CIACRBInTA   = CIACR{InMd1};
CIACRBInCntTA = CIACR{InMd0,InMd1};

```

CONST

```

(* CIAA port A bits *)
CIAGamePort1 = 7; (* gameport 1, pin 6 = fire button *)
CIAGamePort0 = 6; (* gameport 0, pin 6 = fire button *)
CIADiskReady = 5; (* disk ready *)
CIADiskTrack0 = 4; (* disk on track 00 *)
CIADiskProt = 3; (* disk write protect *)
CIADiskChange = 2; (* disk change *)
CIADimLED = 1; (* LED control *)
CIAOverlay = 0; (* memory overlay *)

```

CONST

```

(* CIAA port B bits are the parallel port data *)

```

CONST

```

(* CIAB port A bits *)
CIAComDTR = 7; (* serial: data terminal ready *)
CIAComRTS = 6; (* serial: request to send *)
CIAComCD = 5; (* serial: carrier detect *)
CIAComCTS = 4; (* serial: clear to send *)

```

```
CIAComDSR = 3; (* serial: data set ready *)
CIAPrtSEL = 2; (* printer: select *)
CIAPrtPOUT = 1; (* printer: paper out *)
CIAPrtBUSY = 0; (* printer: busy *)
```

CONST

```
(* CIAB port B bits *)
CIADskMotor = 7; (* disk motor *)
CIADskSEL3 = 6; (* disk select unit 3 *)
CIADskSEL2 = 5; (* disk select unit 2 *)
CIADskSEL1 = 4; (* disk select unit 1 *)
CIADskSEL0 = 3; (* disk select unit 0 *)
CIADskSIDE = 2; (* disk side select *)
CIADskDIREC = 1; (* disk direction of seek *)
CIADskSTEP = 0; (* disk step heads *)
```

VAR

```
CIAA[0BF0001H]: CIAType;
CIAB[0BF0000H]: CIAType;
```

```
END CIAHardware.
```

DEFINITION MODULE CIAResource;

(** -----

Commodore Amiga CIA resource module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 04-Dec-85

Version : 0.00a 04-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;

CONST

CIAName = "ciaa.resource";

CIABName = "ciab.resource";

VAR CIABase: ADDRESS;

PROCEDURE AbleICR(mask: BITSET): BITSET;

(* enable/disable 6526 CIA interrupt control registers.

mask: bit mask indicating interrupts to be modified.
7 IN mask => interrupts disabled, otherwise enabled.

returns: previous enable mask. *)

PROCEDURE AddICRVector(iCRBit: CARDINAL; interrupt: ADDRESS): ADDRESS;

(* attach an interrupt handler to a CIA bit.

iCRBit: Bit to attach interrupt to.
interrupt: address of interrupt handler code.

returns: 0 => successful attach.

otherwise address of owner interrupt code. *)

PROCEDURE RemICRVector(iCRBit: CARDINAL; interrupt: ADDRESS);

(* remove an interrupt handler from a CIA bit.

iCRBit: Bit to attach interrupt to.

interrupt: address of interrupt handler code. *)

PROCEDURE SetICR(mask: BITSET): BITSET;

(* cause, clear, and sample ICR interrupts.

mask: bit mask indicating interrupts to be affected.

7 IN mask => cause interrupts, otherwise reset interrupts.

returns: interrupt register status before making changes. *)

END CIAResource.

DEFINITION MODULE ClipboardDevice;

(** -----

Commodore Amiga clipboard module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 10-Jan-86

Version : 0.00a 10-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE;
FROM Nodes IMPORT Node;
FROM IO IMPORT CmdNonStd, IOStdReq;
FROM Ports IMPORT Message;

CONST

ClipboardName = "clipboard.device";

CONST

CBDPPost = CmdNonStd + 0;
CBDCurrentReadID = CmdNonStd + 1;
CBDCurrentWriteID = CmdNonStd + 2;

TYPE

ClipboardUnitPartial = RECORD

cuNode: Node; (* list of units *)

cuUnitNum: LONGCARD; (* unit number of this unit *)

END;

TYPE

IOClipReq = RECORD

ioClip: IOStdReq;

```

        ioClipID: LONGINT; (* clip identifier *)
    END;

CONST PrimaryClip = 0; (* primary clip unit *)

TYPE
    SatisfyMsg = RECORD
        smMsg: Message;
        smUnit: CARDINAL; (* which clip unit this is *)
        smClipID: LONGINT; (* the clip identifier of the post *)
    END;

TYPE
    ClipStream = RECORD
        csLength: LONGCARD; (* total length of the clip stream *)
        csUnit: CARDINAL; (* clip unit (set by read) *)
    END;

TYPE
    ClipItem = RECORD
        ciLength: LONGCARD; (* total length of the clip item *)
        ciNameLength: CARDINAL; (* clip name length, including null *)
    END;

TYPE
    ClipANSI = RECORD
        caLength: LONGCARD;
    END;

CONST
    BGFColorMap = 1; (* set if the color map is valid *)

TYPE
    ClipBitMap = RECORD
        cbmModes: BITSET; (* graphics viewport modes *)
        cbmXSize: CARDINAL; (* width in pixels *)
        cbmRows: CARDINAL; (* height *)
        cbmDepth: BYTE; (* depth *)
        cbmCMapWords: BYTE; (* number of elements in color map *)
        cbmXOffset: CARDINAL; (* offset of the clip in the source *)
        cbmYOffset: CARDINAL;
        cbmZOffset: BYTE;
        cbmCMapOffset: BYTE;
    END;

```

```
        cbmBitDataLength: LONGCARD;  
    END;
```

```
END ClipboardDevice-
```

DEFINITION MODULE CListLibrary;

(** -----

Commodore Amiga character list module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 84-Dec-85

Version : 0.00a 84-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;

CONST

CListName = "clist.library";

VAR

CListBase: ADDRESS;

TYPE

cListDescriptor = LONGINT;

cPoolDescriptor = ADDRESS;

PROCEDURE AllocCList(cLPool: cPoolDescriptor): cListDescriptor;

(* allocate and initialise a clist.

cLPool: a clist pool that has already been initialised by InitCLPool.

returns: negative => no space available for new clist, otherwise
clist descriptor for clist functions. *)

PROCEDURE ConcatCList(sourceCList, destCList: cListDescriptor): LONGCARD;

(* concatenate two character lists.

sourceCList: clist to be appended to destCList. Empty on return.

destCList: cList onto which sourceCList will be appended.

returns: < 0 => not enough memory for result in destCList,
otherwise OK, *)

PROCEDURE CopyCList(cList: cListDescriptor): cListDescriptor;
(* copy a cList to a new cList - nondestructive.

cList: the cList descriptor for the original cList.

returns: a cList descriptor for the copy, *)

PROCEDURE FlushCList(cList: cListDescriptor);
(* clear a character list.

cList: cList descriptor for erasure, *)

PROCEDURE FreeCList(cList: cListDescriptor);
(* free a cList - release cList descriptor and any resources it uses.

cList: cList descriptor to free, *)

PROCEDURE GetCLBuf(cList: cListDescriptor; buffer: ADDRESS;
maxLength: LONGCARD): LONGCARD;
(* convert a character list to contiguous data.

cList: the cList descripto to convert.

buffer: the address where the data will be stored.

maxLength: the maximum size of the buffer.

returns: the number of characters copied to buffer, *)

PROCEDURE GetCLChar(cList: cListDescriptor): LONGINT;
(* get character from beginning of character list.

cList: cList header from which to get character.

returns: +ve => character at beginnibg of cList;
0FFFFFFx => data unavailable, *)

PROCEDURE GetCLWord(cList: cListDescriptor): LONGINT;

```

(* get word from beginning of character list.

    cList: cList header from which to get character.

    returns: +ve => word at beginning of cList;
             0FFFFxxx => data unavailable. *)

PROCEDURE IncrCLMark(cList: cListDescriptor): LONGCARD;
(* increment a cList mark to next position.

    cList: descrtor of cList to increment mark.

    returns: < 0 => offset not in cList, otherwise OK. *)

PROCEDURE InitCLPool(cLPool: cPoolDescriptor; size: LONGCARD): LONGCARD;
(* initialise a cList pool.

    cLPool: pointer to data area for cList operations.
    size: size of area in characters, < 16MB.

    returns: < 0 => not enough memory to allocate pool, otherwise OK. *)

PROCEDURE MarkCList(cList: cListDescriptor; offset: LONGCARD): LONGCARD;
(* mark a position in a cList.

    cList: cList to mark.
    offset: byte offset into cList, byte 0 = 1st char.

    returns: < 0 => offset not in cList, otherwise OK. *)

PROCEDURE PeekCLMark(cList: cListDescriptor): CHAR;
(* peek at the character in the cList at the mark.

    cList: cList to peek at.

    returns: character at marked position in cList. *)

PROCEDURE PutCLBuf(cList: cListDescriptor; buffer: ADDRESS;
    length: LONGCARD): LONGCARD;
(* append contiguous data onto a character list.

    cList: The cList descriptor for data.

```

buffer: pointer to contiguous data.
length: the number of data bytes in the buffer.

returns: = 0 => OK, otherwise number of characters not added to clist. *)

PROCEDURE PutCLChar(cList: cListDescriptor; char: CHAR): LONGCARD;
(* add a character to the end of a character list.

cList: clist onto which char will be appended.
char: character to append onto end of clist.

returns: = 0 => OK, otherwise character could not be added. *)

PROCEDURE PutCLWord(cList: cListDescriptor; word: CARDINAL): LONGCARD;
(* add a word to the end of a character list.

cList: clist onto which char will be appended.
word: word to append onto end of clist.

returns: = 0 => OK, otherwise number of bytes not added (always 2
as partial words are not added). *)

PROCEDURE SizeCList(cList: cListDescriptor): LONGCARD;
(* gets the number of characters in a character list.

cList: the clist to return the size of.

returns: number of characters in the clist. *)

PROCEDURE SplitCList(cList: cListDescriptor): cListDescriptor;
(* split a clist at the mark.

cList: the clist to split. On return, holds the original clist
upto, but not including, the mark.

returns: -ve: not enough memory to create clist, otherwise a new
clist containing the characters from the mark to the end of
the original clist. *)

PROCEDURE SubCList(cList: cListDescriptor; index: LONGCARD;
length: LONGCARD): cListDescriptor;
(* copy a substring from a clist.

cList: the cList to copy the substring from.
index: where to start copying from, 0 = 1st byte.
length: number of characters to copy.

returns: -ve => not enough room for new cList, otherwise the substring.
if the required substring does not exist then the returned
cList will be less than length characters. *)

PROCEDURE UnGetCLChar(cList: cListDescriptor; char: CHAR): LONGINT;
(* put a character at the beginning of a character list.

cList: the cList to put the character to.
char: the character to put at the beginning of cList.

returns: = 0 => OK, otherwise character cannot be added. *)

PROCEDURE UnGetCLWord(cList: cListDescriptor; word: CARDINAL): LONGINT;
(* put a word at the beginning of a character list.

cList: the cList to put the word to.
word: the word to put at the beginning of cList.

returns: 0 => OK, otherwise number of characters that could not
be added (always 2 as words are never partially added. *)

PROCEDURE UnPutCLChar(cList: cListDescriptor): LONGINT;
(* get a byte from the end of a character list.

cList: the cList to take the character from.

returns: +ve => byte at end of cList;
0FFFFFFxx => data unavailable. *)

PROCEDURE UnPutCLWord(cList: cListDescriptor): LONGINT;
(* get a word from the end of a character list.

cList: the cList to take the word from.

returns: +ve => word at end of cList;
0FFFFxxxx => data unavailable. *)

END CListLibrary.

DEFINITION MODULE Colors;

(** -----
Commodore Amiga colour palette module
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved
----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 21-Jan-86

Version : 0.00a 21-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM Views IMPORT ViewPort;

TYPE

ColorTablePtr = POINTER TO ColorTable;
ColorTable = ARRAY [0..31] OF CARDINAL;

ColorMapPtr = POINTER TO ColorMap;
ColorMap = RECORD
 flags: BYTE;
 type: BYTE; (* = 0 => colorTable is ARRAY count OF CARDINAL *)
 count: CARDINAL; (* nr. entries in colorTable *)
 colorTable: ColorTablePtr;
END;

PROCEDURE FreeColorMap(cMap: ColorMapPtr);

(* free a color map structure and return memory.

cMap: the color map structure to free, allocated by GetColourMap. *)

PROCEDURE GetColorMap(entries: CARDINAL): ColorMapPtr;

(* allocate and initialise a color map.

entries: number of color map entries.

returns: 0 => no memory left, otherwise pointer to the colour map structure. *)

PROCEDURE GetRGB4(cMap: ColorMapPtr; entry: CARDINAL): INTEGER;
(* inquire entry of entry in table.

cMap: the color map to inquire about.

entry: the entry within the colour map to return.

returns: -1 => no valid entry, otherwise a RGB value. *)

PROCEDURE LoadRGB4(VAR vp: ViewPort; cTab: ADDRESS; cnt: CARDINAL);
(* load RGB color values from table.

vp: the viewport to load the new color table into.

cTab: the address of the colour table to load.

cnt: the number of entries in the color table. *)

PROCEDURE SetRGB4(VAR vp: ViewPort; n: CARDINAL; r, g, b: CARDINAL);
(* set one color register for a viewport.

vp: the viewport to set the color of.

n: the color number in the color map.

r,g,b: the red, green, and blue levels for the color. *)

END Colors.

DEFINITION MODULE CommandLine;

(** -----

Commodore Amiga command line extraction module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 19-Dec-85

Version : 0.00a 19-Dec-85 Paul Curtis, TDI.
Original

*)

CONST MaxCLength = 79;

TYPE CLStrings = ARRAY [0..MaxCLength-1] OF CHAR;

PROCEDURE GetCL(VAR argc: CARDINAL; VAR argv: ARRAY OF CLStrings): BOOLEAN;
(* Get command line arguments.

argc: argument count.

argv: arguments with inter-argument padding removed.

returns: TRUE => all arguments read in from the command line,
otherwise more CL arguments were given than could
be fitted into the array.

for example, given the command

Link myprog map sys

the values returned will be:

argc = 4
argv[0] = "Link"
argv[1] = "myprog"
argv[2] = "map"
argv[3] = "sys" -- white space removed.

NOTE: Quotes are not returned surrounding the strings.

If more arguments are given in a command line than available CLString slots, argc will be the maximum number of arguments that were read into the CLStrings; the function will return FALSE.

*)

END CommandLine.

DEFINITION MODULE ConsoleDevice;

```
(** -----  
Commodore Amiga console device module  
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved  
----- **)
```

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 30-Dec-85

Version : 0.00a 30-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;
FROM IO IMPORT CmdMonStd;

CONST

(* console commands *)
CDAskKeyMap = CmdMonStd + 0;
CDSetKeyMap = CmdMonStd + 1;

CONST

(* set graphic rendition (SGR) parameters *)

(* text type *)
SGRPrimary = 0;
SGRBold = 1;
SGRItalic = 3;
SGRUnderscore = 4;
SGRNegative = 7;

(* set foreground colours *)

SGRC1r0 = 30;
SGRC1r1 = 31;
SGRC1r2 = 32;
SGRC1r3 = 33;

```

SGRC1r4 = 34;
SGRC1r5 = 35;
SGRC1r6 = 36;
SGRC1r7 = 37;

```

```

(* set background colours *)

```

```

SGRC1r0BG = 40;
SGRC1r1BG = 41;
SGRC1r2BG = 42;
SGRC1r3BG = 43;
SGRC1r4BG = 44;
SGRC1r5BG = 45;
SGRC1r6BG = 46;
SGRC1r7BG = 47;

```

```

CONST

```

```

(* device status report (DSR) parameters *)

```

```

DSRCPR = 6;

```

```

CONST

```

```

(* SM and RM parameters *)

```

```

MLNM = 20;

```

```

TYPE

```

```

KeyMap = RECORD

```

```

    kmLoKeyMapTypes: ADDRESS;
    kmLoKeyMap: ADDRESS;
    kmLoCapsable: ADDRESS;
    kmLoRepeatable: ADDRESS;
    kmHiKeyMapTypes: ADDRESS;
    kmHiKeyMap: ADDRESS;
    kmHiCapsable: ADDRESS;
    kmHiRepeatable: ADDRESS;

```

```

END;

```

```

TYPE

```

```

KeyQualifier = (SHIFT, CONTROL, ALT, DOWNUP, KS4, KS5, STRING);
KeyQualifierSet = SET OF KeyQualifier;

```

```

CONST

```

```

VanillaKey = KeyQualifierSet{SHIFT,CONTROL,ALT};

```

END ConsoleDevice.

DEFINITION MODULE ConvToString;

(** -----

Commodore Amiga number conversion module

(c) Copyright 1985, 1985 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 24-Dec-85

Version : 0.00a 24-Dec-85 Paul Curtis, TDI.
Original

*)

TYPE Base = [1..36];

PROCEDURE NumberToString(n: LONGCARD; l: CARDINAL; b: Base; fill: CHAR;
neg: BOOLEAN; VAR s : ARRAY OF CHAR);

(*

Conversion of a number to a string of characters.

n is the number to be converted.

l is the minimum length of the generated string.

b is the base of the number.

fill is the character used to pad the string, e.g. "0" no leading
zero suppression, " " leading zero suppression.

neg if TRUE assigns a negative sign to the number.

s is the returned string.

*)

END ConvToString.

DEFINITION MODULE Copper;

(** -----

Commodore Amiga copressor module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 30-Dec-85

Version : 0.00a 30-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT WORD, ADDRESS;

CONST

CopperMove = 0; (* move #n, reg *)

CopperWait = 1; (* wait x,y *)

CopperNextBuf = 2; (* go to next buffer *)

CopperLongFrame = 8000H; (* long frame only *)

CopperShortFrame = 4000H; (* short frame only *)

TYPE

CopListPtr = POINTER TO CopList;

TYPE

CopInsPtr = POINTER TO CopIns;

CopIns = RECORD

CASE opcode: CARDINAL OF

CopperNextBuf: nxtList: CopListPtr; |

CopperWait: VWaitPos: CARDINAL;

HWaitPos: CARDINAL; |

CopperMove: DestAddr: CARDINAL;

DestData: CARDINAL;

```

    END;
END;

```

```

TYPE

```

```

    (* structure of cprlist that points to list that hardware executes *)

```

```

    cprlistptr = POINTER TO cprlist;

```

```

    cprlist = RECORD

```

```

        next: cprlistptr;

```

```

        start: ADDRESS; (* stat of copper list *)

```

```

        max: CARDINAL; (* number of long instructions *)

```

```

    END;

```

```

TYPE

```

```

    CopList = RECORD

```

```

        next: CopListPtr; (* next block for this copper list *)

```

```

        copList: CopListPtr; (* system use *)

```

```

        viewPort: ADDRESS; (* View.ViewPortPtr, system use *)

```

```

        copIns: CopInsPtr; (* start of this block *)

```

```

        copPtr: CopInsPtr; (* intermediate ptr *)

```

```

        copLStart: ADDRESS; (* mrgcop fills this in for Long Frame *)

```

```

        copSStart: ADDRESS; (* mrgcop fills this in for Short Frame *)

```

```

        count: CARDINAL; (* intermediate counter *)

```

```

        MaxCount: CARDINAL; (* max nr. copins for this block *)

```

```

        dyOffset: CARDINAL; (* offset this copper list vertical waits *)

```

```

    END;

```

```

TYPE

```

```

    (* user copper list *)

```

```

    UCopListPtr = POINTER TO UCopList;

```

```

    UCopList = RECORD

```

```

        next: UCopListPtr;

```

```

        firstCopList: CopListPtr; (* head node of this copper list *)

```

```

        copList: CopListPtr; (* node in use *)

```

```

    END;

```

```

TYPE

```

```

    copinitptr = POINTER TO copinit;

```

```

    copinit = RECORD

```

```

        diagstrt: ARRAY [0..3] OF CARDINAL; (*coplist for 1st bitplane*)

```

```

        sprstrtup: ARRAY [0..(2*8*2)+2+(2*2)+2-1] OF CARDINAL;

```

```

        sprstop: ARRAY [0..1] OF CARDINAL;

```

```

    END;

```

```

PROCEDURE CBump(VAR c: UCopList);
  (* move to next copper list instruction.

   c: the copper list to bump. *)

PROCEDURE CMove(VAR c: UCopList; r: ADDRESS; v: WORD);
  (* append copper move instruction to user copper list.

   c: the copper list to append the instruction to.
   r: the register to move to.
   v: the value to move into the register. *)

PROCEDURE CWait(VAR c: UCopList; h, v: CARDINAL);
  (* append copper wait instruction to user copper list.

   c: the copper list to append the instruction to.
   h: the horizontal beam position to wait for.
   v: the vertical beam position to wait for, relative to top of
       viewport. *)

PROCEDURE CopperListInit(VAR c: UCopList);
  (* initialise a user copper list.

   c: the user copper list to initialise. *)

PROCEDURE FreeCopList(VAR copList: CopList);
  (* deallocate intermediate copper list.

   copList: the intermediate copper list to deallocate. *)

PROCEDURE FreeCprList(VAR cprList: cprlist);
  (* deallocate hardware copper list.

   cprList: the hardware copper list to deallocate. *)

END Copper.

```

DEFINITION MODULE CopperUtils;

(** -----

Commodore Amiga coprocessor instruction utilities module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 22-Jan-86

Version : 0.00a 22-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT WORD;
FROM Copper IMPORT UCopList;

PROCEDURE CINIT(VAR c: UCopList);
(* initialise a user copper list.

c: the user copper list to initialise. *)

PROCEDURE CMOVE(VAR c: UCopList; VAR r: ARRAY OF WORD; v: WORD);
(* append copper move instruction to user copper list.

c: the copper list to append the instruction to.

r: the register to move to.

v: the value to move into the register. *)

PROCEDURE CWAIT(VAR c: UCopList; h, v: CARDINAL);
(* append copper wait instruction to user copper list.

c: the copper list to append the instruction to.

h: the horizontal beam position to wait for.

v: the vertical beam position to wait for, relative to top of
viewport. *)

```
PROCEDURE CEND(VAR c: UCopList);  
  (* mark the end of a user copper list. *)  
  
END CopperUtils.
```

DEFINITION MODULE CustomHardware;

(** -----

Commodore Amiga custom hardware definition module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 30-Dec-85

Version : 0.00a 30-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;

TYPE

CustomType = RECORD

bltddat: CARDINAL;
dmaconr: BITSET;
vposr: CARDINAL;
vhposr: CARDINAL;
dskdatr: CARDINAL;
joy0dat: CARDINAL;
joy1dat: CARDINAL;
clxdar: CARDINAL;
adkconr: CARDINAL;
pot0dat: CARDINAL;
pot1dat: CARDINAL;
potinp: CARDINAL;
serdatr: CARDINAL;
dskbytr: CARDINAL;
intelar: CARDINAL;
intreqr: CARDINAL;
dskpt: ADDRESS;
dsklen: CARDINAL;
dskdat: CARDINAL;

```

refptr: CARDINAL;
vposw: CARDINAL;
vhposw: CARDINAL;
copcon: CARDINAL;
serdat: CARDINAL;
serper: CARDINAL;
potgo: CARDINAL;
joytest: CARDINAL;
strequ: CARDINAL;
strvbl: CARDINAL;
strhor: CARDINAL;
strlong: CARDINAL;
bltcon0: CARDINAL;
bltcon1: CARDINAL;
bltafw: CARDINAL;
bltalwm: CARDINAL;
bltcpt: ADDRESS;
bltbpt: ADDRESS;
bltapt: ADDRESS;
bltdpt: ADDRESS;
bltsize: CARDINAL;
pad2d: ARRAY [0..2] OF CARDINAL;
bltcm0d: CARDINAL;
bltbm0d: CARDINAL;
bltam0d: CARDINAL;
bltdm0d: CARDINAL;
pad34: ARRAY [0..3] OF CARDINAL;
bltc0d0t: CARDINAL;
bltb0d0t: CARDINAL;
bltad0t: CARDINAL;
pad3b: ARRAY [0..3] OF CARDINAL;
dsksync: CARDINAL;
cop1lc: LONGCARD;
cop2lc: LONGCARD;
copjmp1: CARDINAL;
copjmp2: CARDINAL;
copins: CARDINAL;
diwstrt: CARDINAL;
diwstop: CARDINAL;
ddfstrt: CARDINAL;
ddfstop: CARDINAL;

```



```

dmacon: BITSET;
clxcon: CARDINAL;
intena: CARDINAL;
intreq: CARDINAL;
adkcon: CARDINAL;
aud: ARRAY [0..3] OF RECORD
    acptr: ADDRESS;
    aclen: CARDINAL;
    acper: CARDINAL;
    acvol: CARDINAL;
    acdat: CARDINAL;
    acpad: ARRAY [0..1] OF CARDINAL;
END;
bplpt: ARRAY [0..5] OF ADDRESS;
pad7c: ARRAY [0..3] OF CARDINAL;
bplcon0: CARDINAL;
bplcon1: CARDINAL;
bplcon2: CARDINAL;
pad83: CARDINAL;
bpl1mod: CARDINAL;
bpl2mod: CARDINAL;
pad86: ARRAY [0..1] OF CARDINAL;
bpldat: ARRAY [0..5] OF CARDINAL;
pad8e: ARRAY [0..1] OF CARDINAL;
sprpt: ARRAY [0..7] OF ADDRESS;
spr: ARRAY [0..7] OF RECORD
    pos: CARDINAL;
    ctl: CARDINAL;
    dataa: CARDINAL;
    datab: CARDINAL;
END;
color: ARRAY [0..31] OF CARDINAL;
END;

```

VAR

Custom[0DFF000H]: CustomType; (* custom hardware chips *)

END CustomHardware.

DEFINITION MODULE Devices;

(** -----

Commodore Amiga devices module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 16-Dec-85

Version : 0.00a 16-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, WORD, ADDRESS;
FROM Ports IMPORT MsgPortPtr;
FROM Libraries IMPORT Library;

TYPE

DevicePtr = POINTER TO Device;
Device = RECORD
 ddLibrary: Library;
END;

UnitStates = (UnitActive, UnitInTask);

UnitStateSet = SET OF UnitStates;

UnitPtr = POINTER TO Unit;
Unit = RECORD

 unitMsgPort: MsgPortPtr; (* queue for unprocessed messages *)
 unitFlags: UnitStateSet;
 unitPad: BYTE;
 unitOpenCnt: CARDINAL; (* number of active opens *)
END;

PROCEDURE AddDevice(device: DevicePtr);

(* add a device to the system.

device: pointer to an initialised device node. *)

PROCEDURE CloseDevice(VAR ioRequest: ARRAY OF WORD);

(* conclude access to a device.

ioRequest: the IO request structure returned by OpenDevice. *)

PROCEDURE OpenDevice(VAR devName: ARRAY OF CHAR; unitNum: LONGCARD;

VAR ioRequest: ARRAY OF WORD; flags: LONGCARD): LONGCARD;

(* gain access to a device.

devName: the name of the device to open.

unitNum: the unit number to open on that device - device specific.

ioRequest: the IO request block to return with various fields initialised.

flags: device driver specific flags.

returns: 0 => no error, otherwise error number. *)

PROCEDURE RemDevice(device: DevicePtr): LONGCARD;

(* remove a device from the system.

dev: pointer to a device node.

returns: 0 => success, otherwise error number. *)

END Devices.

DEFINITION MODULE DiskFontLibrary;

(** -----

Commodore Amiga disk font module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 10-Jan-86

Version : 0.00a 10-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM Text IMPORT TextFont, TextAttr;
FROM Nodes IMPORT Node;

CONST

DiskFontName = "diskfont.library";

VAR

DiskFontBase: ADDRESS;

CONST

MaxFontPath = 256;

TYPE

FontContents = RECORD

fcFileName: ARRAY [0..MaxFontPath-1] OF CHAR;

fcYSize: CARDINAL;

fcStyle: BYTE;

fcFlags: BYTE;

END;

CONST

FCHID = 0F00H;

DFHID = 0F80H;

TYPE

FontContentsHeaderPtr = POINTER TO FontContentsHeader;

FontContentsHeader = RECORD

 fchFileID: CARDINAL; (* set to FCHID *)

 fchNumEntries: CARDINAL; (* nr. FontConts entries *)
END;

CONST

 MaxFontName = 32;

TYPE

DiskFontHeaderPtr = POINTER TO DiskFontHeader;

DiskFontHeader = RECORD

 dfhDF: Node; (* node to link disk fonts *)

 dfhFileID: CARDINAL; (* set to DFHID *)

 dfhRevision: CARDINAL; (* font revision *)

 dfhSegment: ADDRESS; (* segment address when loaded *)

 dfhName: ARRAY [0..MaxFontName] OF CHAR;

 dfhTF: TextFont; (* loaded TextFont structure *)

END;

CONST

 AFMemory = 0;

 AFDisk = 1;

TYPE

AvailFont = RECORD

 afType: BITSET; (* memory or disk *)

 afAttr: TextAttr; (* text attributes for font *)
END;

TYPE

AvailFontsHeaderPtr = POINTER TO AvailFontsHeader;

AvailFontsHeader = RECORD

 afhNumEntries: CARDINAL; (* nr. AvailFont entries *)

 afhAvailFonts: ARRAY [0..0] OF AvailFont;

END;

PROCEDURE AvailFonts(buffer: ADDRESS; bufLen: LONGCARD; typ: BITSET): LONGCARD;
(* build an array of all fonts in memory / on disk.

buffer: the place where the font header and entries will be stored.
buflen: the length of the buffer in bytes.
typ: the types to search for, e.g.
 {AFMemory} - search for memory fonts only.
 {AFMemory, AFDisk} - search for memory and disk fonts.

returns: 0 => no error, otherwise the number of bytes needed in
 addition to those supplied; in this case the buffer
 was not filled. *)

PROCEDURE OpenDiskFont(VAR textAttr: TextAttr): LONGCARD;
(* load and get a pointer to a disk font.

textAttr: the font attributed desired.

returns: 0 => font not loaded, otherwise a font descriptor. *)

END DiskFontLibrary.

DEFINITION MODULE DMABits;

(** -----

Commodore Amiga DMA control bit definition module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 20-Jan-86

Version : 0.00a 20-Jan-86 Paul Curtis, TDI.
Original

*)

CONST

(* write definitions for dmaconw *)

DMASetClr = 15;
DMAAud0 = 0;
DMAAud1 = 1;
DMAAud2 = 2;
DMAAud3 = 3;
DMADisk = 4;
DMASprite = 5;
DMABlitter = 6;
DMACopper = 7;
DMARaster = 8;
DMAMaster = 9;
DMABlitHog = 10;

CONST

(* read definitions for dmaconr *)

DMABlitDone = 14;
DMABlitNZer = 13;

END DMABits.

DEFINITION MODULE DMAutils;

(** -----

Commodore Amiga DMA utilities module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 20-Jan-86

Version : 0.00a 20-Jan-86 Paul Curtis, TDI.
Original

*)

PROCEDURE DisplayOn;

(* turn display DMA on. *)

PROCEDURE DisplayOff;

(* turn display DMA off. *)

PROCEDURE SpriteOn;

(* turn sprite display on. *)

PROCEDURE SpriteOff;

(* turn sprite display on. *)

END DMAutils.

DEFINITION MODULE DOSCodeLoader;

(** -----

Commodore Amiga DOS code loading module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 31-Dec-85

Version : 0.00a 31-Dec-85 Paul Curtis, TDI.
Split from old DOS module.

*)

FROM SYSTEM IMPORT ADDRESS;

FROM DOSFiles IMPORT FileHandle;

PROCEDURE Execute(VAR cmdString: ARRAY OF CHAR;
input, output: FileHandle): BOOLEAN;

(* Execute a CLI command.

cmdString: the CLI string to execute.

input: filehandle: 0 => execute command string and return.

<> 0 => execute command string, then take
commands from the input file until the end
of file.

output: filehandle: 0 => output to current window.

<> 0 => redirect output to the output file. *)

PROCEDURE LoadSeg(VAR name: ARRAY OF CHAR): ADDRESS;

(* Load a load module into memory.

name: the name of the load module (on disk) to load into memory.

returns: 0 => error occurred, otherwise the code was correctly
loaded into memory and the returned value is a pointer
to the start of the segment list. To execute the code,

see CreateProc. *)

PROCEDURE UnloadSeg(segment: ADDRESS);

(* Unload a segment previously loaded by LoadSeg.

segment: pointer to segment, returned by LoadSeg. *)

END DOSCodeLoader.

DEFINITION MODULE DOSExtensions;

(** -----

Commodore Amiga DOS extended data types module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 16-Jan-86

Version : 0.00a 16-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;
FROM DOSLibrary IMPORT BPTR, BSTR, DateStampRec;
FROM Libraries IMPORT Library;
FROM Ports IMPORT MsgPortPtr, MsgPort, Message;
FROM Tasks IMPORT Task;

TYPE

(* structure of all DOS processes.

Create and DeviceProc returns pointer to the MsgPort in this structure.
To get the correct process pointer use:

DevProc = ProcessPtr(ADDRESS(DeviceProc()) - ADDRESS(TSIZE(Task))); *)

ProcessPtr = POINTER TO Process;

Process = RECORD

prTask: Task;
prMsgPort: MsgPort;
prPad: CARDINAL;
prSegList: BPTR; (* array of seg lists used by this process *)
prStackSize: LONGCARD; (* size of process stack, in bytes *)
prGlobVec: ADDRESS; (* global vector for this process *)
prTaskNum: LONGCARD; (* 0 => not a CLI task *)
prStackBase: BPTR; (* high memory end of process stack *)
prResult2: LONGINT; (* secondary result from last call *)

```

prCurrentDir: BPTR; (* lock associated with current directory *)
prCIS: BPTR; (* current CLI input stream *)
prCOS: BPTR; (* current CLI output stream *)
prConsoleTask: ADDRESS; (* console handler for current window *)
prFileSystemTask: ADDRESS; (* file handler for current drive *)
prCLI: BPTR; (* pointer to ConsoleLineInterpreter *)
prReturnAddr: ADDRESS; (* pointer to previous stack frame *)
prPktWait: ADDRESS; (* function to be called when awaiting msg *)
prWindowPtr: ADDRESS; (* window for error printing *)
END;

```

TYPE

(* a BPTR is returned to this structure by Open() *)

FileHandleBlock = RECORD

```

    fhLink: ADDRESS; (* EXEC message *)
    fhPort: MsgPortPtr; (* reply port for the packet *)
    fhType: MsgPortPtr; (* port to do PutMsg() to *)
    fhBuf: BPTR;
    fhPos: LONGCARD;
    fhEnd: LONGCARD;
    fhFuncs: PROCEDURE;
    fhFunc2: PROCEDURE;
    fhFunc3: PROCEDURE;
    fhArgs: LONGCARD;
    fhArg2: LONGCARD;
END;

```

TYPE

DosPacketPtr = POINTER TO DosPacket;

DosPacket = RECORD

```

    dpLink: ADDRESS; (* EXEC message *)
    dpPort: MsgPortPtr; (* reply port for the packet *)
    dpType: LONGCARD; (* see below *)
    dpRes1: LONGCARD; (* function result *)
    dpRes2: LONGCARD; (* IoErr result *)
    dpArg1: LONGCARD;
    dpArg2: LONGCARD;
    dpArg3: LONGCARD;
    dpArg4: LONGCARD;
    dpArg5: LONGCARD;
    dpArg6: LONGCARD;

```

```

        dpArg7: LONGCARD;
    END;

```

```

TYPE

```

```

    StandardPacket = RECORD
        spMsg: Message;
        spPkt: Message;
    END;

```

```

CONST

```

```

    (* Packet types *)
    ActionNIL           = 0;
    ActionGetBlock      = 2;
    ActionSetMap        = 4;
    ActionDie           = 5;
    ActionEvent         = 6;
    ActionCurrentVolume = 7;
    ActionLocateObject  = 8;
    ActionRenameDisk    = 9;
    ActionWrite         = 57H; (* "W" *)
    ActionRead          = 52H; (* "R" *)
    ActionFreeLock      = 15;
    ActionDeleteObject  = 16;
    ActionRenameObject  = 17;
    ActionCopyDir       = 19;
    ActionWaitChar      = 20;
    ActionSetProtect    = 21;
    ActionCreateDir     = 22;
    ActionExamineObject = 23;
    ActionExamineNext   = 24;
    ActionDiskInfo      = 25;
    ActionInfo          = 26;
    ActionSetComment    = 28;
    ActionParent        = 29;
    ActionTimer         = 30;
    ActionInhibit       = 31;
    ActionDiskType      = 32;
    ActionDiskChange    = 33;

```

```

TYPE

```

```

    DosBaseRec = RECORD

```

```

        dllib: Library;
        dlRoot: POINTER TO RootNode;
        dlGV: ADDRESS; (* pointer to BCPL global vector *)
        dlA2: ADDRESS; (* private register dump place for DOS *)
        dlA5: ADDRESS;
        dlA6: ADDRESS;
    END;

```

TYPE

RootNode = RECORD

```

        rnTaskArray: BPTR; (* [0] is max number of CLI's
                               [1] is address of process id of CLI 1
                               [n] is address of process id of CLI n *)
        rnConsoleSegment: BPTR; (* SegList for the CLI *)
        rnTime: DateStampRec; (* current time *)
        rnRestartSeg: BPTR; (* SegList for the disk validator process *)
        rnInfo: BPTR; (* pointer to the Info structure *)
    END;

```

TYPE

DosInfo = RECORD

```

        diMcName: BPTR; (* network name of this machine; currently 0 *)
        diDevInfo: BPTR; (* DeviceList *)
        diDevices: BPTR; (* zero *)
        diHandlers: BPTR; (* zero *)
        diNetHand: BPTR; (* network handler processid; currently zero *)
    END;

```

TYPE

(* DOS Processes started from the CLI (by RUN or NEWCLI) have this data associated with them *)

CommandLineInterfacePtr = POINTER TO CommandLineInterface;

CommandLineInterface =

RECORD

```

        cliResult2: LONGINT; (* IoErr from last command *)
        cliSetName: BPTR; (* name of current directory *)
        cliCommandDir: BPTR; (* lock associated with command directory *)
        cliReturnCode: LONGINT; (* return code from last command *)
        cliCommandName: BPTR; (* a BSTR; name of current command *)
        cliFailLevel: LONGINT; (* fail level, set by FAILAT *)
        cliPrompt: BSTR; (* current prompt, set by PROMPT *)

```

```

cliStandardInput: BPTR; (* default CLI input *)
cliCurrentInput: BPTR; (* current CLI input *)
cliCommandFile: BSTR; (* name of EXECUTE command file *)
cliInteractive: LONGINT; (* <> 0 => prompts required *)
cliBackground: LONGINT; (* <> 0 => CLI created by RUN *)
cliCurrentOutput: BPTR; (* current CLI output *)
cliDefaultStack: LONGINT; (* stack size to be obtained in long words *)
cliStandardOutput: BPTR; (* default CLI output *)
cliModule: BPTR; (* SegList of currently loaded command *)
END;

```

TYPE

```

DeviceList = RECORD
    dlNext: BPTR; (* next device list *)
    dlType: LONGINT;
    dlTask: MsgPortPtr; (* ptr to handler task *)
    dlLock: BPTR;
    dlVolumeDate: DateStampRec; (* creation date *)
    dlLockList: BPTR; (* outstanding locks *)
    dlDiskType: ARRAY [0..3] OF CHAR; (* "DOS", etc. *)
    dlunused: LONGINT;
    dlName: POINTER TO BSTR; (* bptr to bcpl name *)
END;

```

CONST

```

(* dlTypes *)
DLTDevice    = 0;
DLTDirectory = 1;
DLTVolume    = 2;

```

TYPE

```

(* a lock structure, as returned by Lock() or DupLock() *)
FileLockBlock = RECORD
    flLink: BPTR; (* next lock *)
    flKey: LONGINT; (* disk block number *)
    flAccess: LONGINT; (* exclusive or shared *)
    flTask: MsgPortPtr; (* handler task's port *)
    flVolume: BPTR; (* bptr to a DeviceList *)
END;

```

END DOSExtensions.

DEFINITION MODULE DOSFiles;

(** -----
Commodore Amiga DOS file handling module
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved
----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 31-Dec-85

Version : 0.00a 31-Dec-85 Paul Curtis, TDI.
Split from old DOS module.

*)

FROM SYSTEM IMPORT ADDRESS;
FROM DOSLibrary IMPORT DateStampRec, BSTR;

TYPE

FileLock = LONGCARD; (* a file lock *)
FileHandle = LONGCARD; (* a file handle *)

CONST

(* File modes *)
ModeOldFile = 1005; (* open existing file for read or write *)
ModeNewFile = 1006; (* delete old file, create new one *)

CONST

(* Seek modes *)
OffsetBeginning = -1; (* relative to beginning of file *)
OffsetCurrent = 0; (* relative to current file position *)
OffsetEnd = 1; (* relative to end of file *)

CONST

(* Lock types *)
SharedLock = -2; (* file may be read by others *)
ExclusiveLock = -1; (* no other access allowed *)
AccessRead = SharedLock;
AccessWrite = ExclusiveLock;

CONST

(* protection bits, used in FileInfoBlock.fibProtection *)

FIBRead = 3;

FIBWrite = 2;

FIBExecute = 1;

FIBDelete = 0;

TYPE

ProtMask = SET OF [0..31];

TYPE

(* LONG ALIGNED *)

FileInfoBlock = RECORD

fibDiskKey : LONGCARD;

fibDirEntryType : LONGCARD; (* < 0 => plain file, > 0 => directory *)

fibFileName : ARRAY [0..107] OF CHAR; (* filename, term = 0C *)

fibProtection : ProtMask; (* protection *)

fibEntryType : LONGCARD;

fibSize : LONGCARD; (* nr. bytes in file *)

fibNumBlocks : LONGCARD; (* nr. blocks in file *)

fibDate : DateStampRec; (* last change time of file *)

fibComment : ARRAY [0..115] OF CHAR; (* file comment, term = 0C *)

END;

CONST

(* Disk states returned in InfoData.idDiskState *)

IDWriteProtected = 80; (* disk has write protect tab on *)

IDValidating = 81; (* disk is currently being validated *)

IDValidated = 82; (* disk is consistent and writeable *)

TYPE

(* LONG ALIGNED *)

InfoData = RECORD

idNumSoftErrors: LONGCARD; (* number of soft errors on disk *)

idUnitNumber: LONGCARD; (* which unit disk is, or was, mounted on *)

idDiskState: LONGCARD;

idNumBlocks: LONGCARD; (* nr. blocks on disk *)

idNumBlocksUsed: LONGCARD; (* nr. blocks in use *)

idBytesPerBlock: LONGCARD; (* nr. bytes in one disk block *)

idDiskType: LONGCARD; (* disk type code *)

idVolumeNode: BSTR; (* BCPL pointer to volume name *)

idInUse: LONGCARD; (* 0 => not in use *)

END;

CONST

(* Disk types *)

IDNoDiskPresent = -1;

IOUnreadableDisk = 42414400H; (*BAD *)

IODosDisk = 444F5300H; (*DOS *)

IDNotReallyDos = 4E444F53H; (*NDOS*)

IDKickstartDisk = 4B49434BH; (*KICK*)

PROCEDURE Close(file: FileHandle);

(* Close an open file.

file: the filehandle returned by Open that you wish to close. *)

PROCEDURE CreateDir(VAR name: ARRAY OF CHAR): FileLock;

(* Create a new directory.

name: the name of the new directory to create, null terminated.

returns: 0 => cannot create directory, otherwise returns a
shared read lock. *)

PROCEDURE CurrentDir(lock: FileLock): FileLock;

(* Make directory associated with a lock the current directory.

lock: the directory lock that should be made the current directory.

returns: the old lock. 0 => directory is the root of the startup disk. *)

PROCEDURE DeleteFile(VAR name: ARRAY OF CHAR): BOOLEAN;

(* Delete a file or directory.

name: the name of the file/directory to delete, null terminated.

returns: TRUE => successful deleteion, error otherwise. *)

PROCEDURE DupLock(lock: FileLock): FileLock;

(* Duplicate a lock.

lock: the shared filing system READ lock that should be duplicated.

returns: a shared read lock to the same item. *)

PROCEDURE Examine(lock: FileLock; VAR fib: FileInfoBlock): BOOLEAN;
(* Examine a directory for file associated with a lock.

lock: the lock for the file.

fib: the FIB filled with file information, LONG ALIGNED.

returns: TRUE => ok, fib is valid, otherwise error. *)

PROCEDURE ExNext(lock: FileLock; VAR fib: FileInfoBlock): BOOLEAN;
(* Examine the next entry in the directory.

lock: the lock for the file.

fib: the FIB filled with file information, LONG ALIGNED.

returns: TRUE => ok, fib is valid, otherwise error. *)

PROCEDURE Info(lock: FileLock; VAR infoData: InfoData): BOOLEAN;
(* Return information about a disk.

lock: a file lock for the disk or an file on the disk.

infoData: the filled data for the disk, LONG ALIGNED.

returns: TRUE => ok, infoData is valid, otherwise error. *)

PROCEDURE Input(): FileHandle;
(* Return initial input filehandle.

returns: initial input file handle. *)

PROCEDURE IoErr(): LONGINT;
(* Return more information on an error.

returns: an error number, listed above. *)

PROCEDURE IsInteractive(file: FileHandle): BOOLEAN;
(* Discover if a file is connected to a virtual terminal or not.

file: the filehandle for the interactive inquiry.

returns: TRUE => connected to a virtual terminal, otherwise FALSE. *)

PROCEDURE Lock(VAR name: ARRAY OF CHAR; accessMode: LONGCARD): FileLock;
(* Lock a directory or file.

name: the name of the file or directory to lock, null terminated.

accessMode: AccessRead => shared read lock,

AccessWrite => exclusive write lock.

returns: 0 => cannot obtain filing system lock (fail), otherwise
returns a lock. *)

PROCEDURE Open(VAR name: ARRAY OF CHAR; accessMode: LONGINT): FileHandle;
(* Open a file for input or output.

name: name of file to open, null terminated.

accessMode: ModeNewFile or ModeOldFile.

returns: 0 => cannot open file, otherwise file is open and
returns the file handle. *)

PROCEDURE Output(): FileHandle;
(* Return initial output filehandle.

returns: initial output file handle. *)

PROCEDURE ParentDir(lock: FileLock): FileLock;
(* Obtain parent of directory or file.

lock: the lock of the file to obtain the parent of.

returns: 0 => root of current filing system, otherwise
the lock of the parent. *)

PROCEDURE Read(file: FileHandle; Buffer: ADDRESS; length: LONGCARD): LONGINT;
(* Read bytes of data from file.

file: the filehandle for the file to read the data from.

buffer: pointer to where the data will be put.

length: the number of bytes to read into the buffer (the buffer

length.)

returns: -1 => error, otherwise number of bytes read. *)

PROCEDURE Rename(VAR oldName, newName: ARRAY OF CHAR): BOOLEAN;
(* Rename a directory or file.

oldName: filename to rename from, null terminated.

newName: filename to rename to, null terminated.

returns: TRUE => rename ok, otherwise not renamed. *)

PROCEDURE Seek(file: FileHandle; position: LONGINT; mode: LONGINT): LONGINT;
(* Move to logical position in a file.

file: the filehandle of the file to position.

position: the offset to move the cursor.

mode: One of the seek modes above.

returns: -1 => error, otherwise the old file position. *)

PROCEDURE SetComment(VAR name, comment: ARRAY OF CHAR): BOOLEAN;
(* Set the comment field of a file.

name: the name of the file to attach the comment to, null terminated.

comment: the comment string, < 80 characters, null terminated.

returns: TRUE => comment set ok, otherwise comment not set. *)

PROCEDURE SetProtection(VAR name: ARRAY OF CHAR; prot: ProtMask): BOOLEAN;
(* Set file or directory protection.

name: the name of the file to set the protection of, null terminated.

prot: the protection mask, protection bits described above.

returns: TRUE => protection set successful, otherwise protection
not set. *)

PROCEDURE Unlock(lock: FileLock);
(* Unlock a directory or file.

lock: the lock of the file/directory to unlock. *)

```

PROCEDURE WaitForChar(file: FileHandle; timeOut: LONGCARD): BOOLEAN;
  (* Indicate if characters arrive within a timeout period or not.

  file: the file that characters are expected to arrive from.
  timeOut: the timeout period, in microseconds.

  returns: TRUE => character arrived within timeout, otherwise
           character did not arrive. *)

PROCEDURE Write(file: FileHandle; buffer: ADDRESS; length: LONGCARD): LONGINT;
  (* Write bytes of data to file.

  file: the filehandle for the file to write the data to.
  buffer: pointer to where the data will be found.
  length: the number of bytes to write from buffer to the file.

  returns: -1 => error, otherwise number of bytes written. *)

END DOSFiles.

```

DEFINITION MODULE DOSLibrary;

(** -----

Commodore Amiga disc operating system, base module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 12-Dec-85

Version : 0.00b 31-Dec-85 Paul Curtis, TDI.
Spilt code loading and process handling
up into separate modules.
0.00a 12-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;

CONST

DOSName = "dos.library";

VAR

DOSBase: ADDRESS;

TYPE

BPTR = ADDRESS; (* a BCPL pointer *)

BSTR = BPTR; (* a BCPL string: e.g. "M2C" would be held in bytes as
03 4D 32 43, where 3 = length of string *)

CONST

TicksPerSecond = 50; (* nr. ticks in one second *)

TYPE

DateStampRec = RECORD

dsDays: LONGCARD; (* nr. days since 01-Jan-78 *)

dsMinute: LONGCARD; (* nr. minutes past midnight *)

```

dsTick: LONGCARD; (* nr. ticks past minute *)
END;

```

```
CONST
```

```
(* Errors *)
```

```

ErrorNoFreeStore      = 183;
ErrorNoDefaultDir     = 201;
ErrorObjectInUse      = 202;
ErrorObjectExists     = 203;
ErrorDirNotFound      = 204;
ErrorObjectNotFound   = 205;
ErrorBadStreamName    = 206;
ErrorObjectTooLarge   = 207;
ErrorActionNotKnown   = 209;
ErrorInvalidComponentName = 210;
ErrorInvalidLock      = 211;
ErrorObjectWrongType  = 212;
ErrorDiskNotValidated = 213;
ErrorDiskWriteProtected = 214;
ErrorRenameAcrossDevices = 215;
ErrorDirectoryNotEmpty = 216;
ErrorTooManyLevels    = 217;
ErrorDeviceNotMounted = 218;
ErrorSeekError        = 219;
ErrorCommentTooBig    = 220;
ErrorDiskFull         = 221;
ErrorDeleteProtected  = 222;
ErrorWriteProtected   = 223;
ErrorReadProtected    = 224;
ErrorNotADOSDisk     = 225;
ErrorNoDisk           = 226;
ErrorNoMoreEntries    = 232;

```

```
CONST
```

```
(* AmigaDOS conventional return codes *)
```

```

ReturnOK    = 0; (* no problems *)
ReturnWarn  = 5; (* warning *)
ReturnError = 10; (* something went wrong *)
ReturnFail  = 20; (* complete failure to perform *)

```

```
CONST
```

```
(* Bit numbers for breaks *)
```



```
SIGBreakC = 12;  
SIGBreakD = 13;  
SIGBreakE = 14;  
SIGBreakF = 15;
```

```
END DOSLibrary.
```

DEFINITION MODULE DOSProcessHandler;

(** -----
Commodore Amiga DOS process handling module
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved
----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 31-Dec-85

Version : 0.00a 31-Dec-85 Paul Curtis, TDI.
Spilt from old DOS module.

*)

FROM SYSTEM IMPORT ADDRESS;
FROM DOSLibrary IMPORT DateStampRec;

TYPE ProcessID = ADDRESS; (* a process identifier *)

PROCEDURE CreateProc(VAR name: ARRAY OF CHAR; pri: LONGINT;
segment: ADDRESS; stackSize: LONGCARD): ProcessID;
(* Create a new process.

name: name of new process, null terminated.
pri: priority of new process.
segment: segment list. See LoadSeg, UnloadSeg.
stackSize: the stack size of the new task.

returns: 0 => cannot create process, otherwise the process
identifier of the new process. *)

PROCEDURE DateStamp(VAR dateStamp: DateStampRec);
(* Obtain the date and time in internal format.

dateStamp: returns the current date and time in internal format.
If all three elements of the stamp are zero, date and
time is not set. *)

PROCEDURE Delay(timeout: LONGCARD);

(* Delay a process for a specified time.

timeout: the number of ticks for the calling process to wait,
one tick being 1/50th of a second. *)

PROCEDURE DeviceProc(VAR name: ARRAY OF CHAR): ProcessID;

(* Returns the process identifier of the device that handles that IO.

name: the name of the device for which the process identifier
should be returned.

returns: 0 => cannot find a process handler, otherwise the
process identifier of the device handler. If name refers
to a file on a mounted device, then IoErr returns a
directory lock. *)

PROCEDURE Exit(returnCode: LONGCARD);

(* Exit from a program or process.

returnCode: If program run under CLI, then the value that should
be returned to CLI. If exiting from a process,
Exit releases the space associated with the stack,
segment list, and process structure. *)

END DOSProcessHandler.

DEFINITION MODULE Exec;

(** -----
Commodore Amiga ROM executive module
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved
----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 18-Jan-86

Version : 0.00a 18-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;

CONST

ExecName = "exec.library"; (* NOTE: ExecBase is exported from AMIGAX *)

PROCEDURE Debug;

(* enter ROM debugger *)

PROCEDURE Alert(alertNum: LONGCARD; parameters: LONGCARD);

END Exec.

DEFINITION MODULE ExecBase;

(** -----

Commodore Amiga EXEC data definition module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 14-Jan-86

Version : 0.00a 14-Jan-86 Paul Curtis, TDI,
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM Interrupts IMPORT SoftIntList, IntVector;
FROM Libraries IMPORT Library;
FROM Tasks IMPORT TaskPtr;
FROM Lists IMPORT List;

TYPE

ExecBasePtr = POINTER TO RECORD

LibNode: Library;

SoftVer: CARDINAL; (* soft revision of EXEC *)

LowMemChkSum: INTEGER;

ChkBse: LONGCARD; (* system base pointer complement *)

ColdCapture: PROCEDURE; (* coldstart soft vector *)

CoolCapture: PROCEDURE;

WarnCapture: PROCEDURE;

SysStkUpper: ADDRESS; (* system stack upper bound *)

SysStkLower: ADDRESS; (* system stack lower bound *)

MaxLocMem: LONGCARD;

DebugEntry: ADDRESS;

DebugData: ADDRESS;

AlertData: ADDRESS;

RsvdExt: ADDRESS;

```

ChkSum: CARDINAL;

IntVects: ARRAY [0..15] OF IntVector;

ThisTask: TaskPtr; (* current task *)
IdleCount: LONGCARD; (* idle counter *)
DispCount: LONGCARD; (* dispatch counter *)
Quantum: CARDINAL; (* time slice quantum *)
Elapsed: CARDINAL; (* current quantum ticks *)
SysFlags: BITSET; (* system flags *)
IDNestCnt: BYTE; (* interrupt disable nesting count *)
TDNestCnt: BYTE; (* task disable nesting count *)

AttnFlags: BITSET; (* interrupt attention *)
AttnResched: BITSET; (* rescheduling attention *)
ResModules: ADDRESS; (* resident module array pointer *)

TaskTrapCode: PROCEDURE;
TaskExceptCode: PROCEDURE;
TaskExitCode: PROCEDURE;
TaskSigAlloc: LONGCARD;
TaskTrapAlloc: LONGCARD;

MemList: List;
ResourceList: List;
DeviceList: List;
IntrList: List;
LibList: List;
PortList: List;
TaskReady: List;
TaskWait: List;

SoftInts: ARRAY [0..5] OF SoftIntList;

LastAlert: ARRAY [0..3] OF LONGINT;

ExecBaseReserved: ARRAY [0..8] OF LONGINT;

END;

```

END ExecBase.

DEFINITION MODULE Gadgets ;

```
(* -----  
    TDI Modula-2/Amiga : Intuition/Gadgets  
----- *)  
  
(* ----- *)  
(* (c) Copyright TDI Software Ltd. 1986. All Rights Reserved *)  
(* ----- *)
```

FROM SYSTEM IMPORT ADDRESS ;
FROM Intuition IMPORT IntuitionText, Gadget, GadgetFlags, GadgetFlagsSet,
Requester, PropFlagsSet ;
(* NB. The Intuition library must be loaded before calling this module
(see Intuition.def). *)

CONST
(* Compound flags *)

HighBits = GadgetFlagsSet{Ga0,Ga1} ;
HighComplement = GadgetFlagsSet{} ; (* Complement the select box *)
HighBox = GadgetFlagsSet{Ga0} ; (* Draw a box around the image *)
HighImage = GadgetFlagsSet{Ga1} ; (* Alternate image *)
HighNone = GadgetFlagsSet{Ga0,Ga1} ; (* don't highlight *)

CONST

(* Gadget types *)

(* These are the Gadget Type definitions for the variable GadgetType
gadget number type MUST start from one. NO TYPES OF ZERO ALLOWED.
first comes the mask for Gadget flags reserved for Gadget typing *)

GadgetType = 0FC00H ; (* all Gadget Global Type flags (padded) *)
SysGadget = 08000H ; (* 1 = SysGadget, 0 = AppliGadget *)
ScreenGadget = 04000H ; (* 1 = ScreenGadget, 0 = WindowGadget *)
GZZGadget = 02000H ; (* 1 = Gadget for GIMMEZEROZERO borders *)
RequestorGadget = 01000H ; (* 1 = this is a Requester Gadget *)

(* system gadgets *)
Sizing = 00010H ;
WindowDragging = 00020H ;

```

ScreenDragging = 00030H ;
WindowUpFront = 00040H ;
ScreenUpFront = 00050H ;
WindowDownBack = 00060H ;
ScreenDownBack = 00070H ;
Close          = 00080H ;
(* application gadgets *)
BoolGadget     = 00001H ;
Gadget0002     = 00002H ;
PropGadget     = 00003H ;
StrGadget      = 00004H ;

```

```

PROCEDURE AddGadget ( Pointer : ADDRESS ; VAR Gad : Gadget ;
                      Position : INTEGER ) : INTEGER ;
(* Add gadget to window or screen pointed to. Returns position gadget was
   added. *)

```

```

PROCEDURE ModifyProp ( VAR Gad : Gadget ; Pointer : ADDRESS ;
                      VAR Req : Requester ; Flags : PropFlagsSet ;
                      HorizPot, VertPot, HorizBody, VertBody : CARDINAL ) ;
(* Modify the parameters of a proportional gadget *)

```

```

PROCEDURE OffGadget ( VAR Gad : Gadget ; Pointer : ADDRESS ;
                     VAR Req : Requester ) ;
(* Disables the specified gadget. *)

```

```

PROCEDURE OnGadget ( VAR Gad : Gadget ; Pointer : ADDRESS ;
                    VAR Req : Requester ) ;
(* Enables the specified gadget. *)

```

```

PROCEDURE RefreshGadgets ( VAR Gad : Gadget ; Pointer : ADDRESS ;
                           VAR Req : Requester ) ;
(* Refreshes all of the gadgets in the gadget list *)

```

```

PROCEDURE RemoveGadget ( Pointer : ADDRESS ; VAR Gad : Gadget ) : INTEGER ;
(* Removes the given gadget. Returns the original position of the gadget *)

```

```

END Gadgets.

```


DEFINITION MODULE GamePortDevice;

(** -----

Commodore Amiga game port device module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 29-Jan-86

Version : 0.00a 29-Jan-86 Paul Curtis, TDI.
Original

*)

FROM IO IMPORT CmdNonStd;

CONST

GamePortName = "gamport.device";

CONST

GPReadEvent = CmdNonStd + 0;

GPAskCType = CmdNonStd + 1;

GPSetCType = CmdNonStd + 2;

GPAskTrigger = CmdNonStd + 3;

GPSetTrigger = CmdNonStd + 4;

CONST

GPErrSetCType = 1; (* this controller not valid at this time *)

CONST

(* key transition triggers *)

DownKeys = 0;

UpKeys = 1;

TYPE

GamePortTrigger = RECORD

gptKeys: BITSET; (* key transition triggers *)

gptTimeout: CARDINAL; (* time trigger, vert blanks *)

```
    gptXDelta: CARDINAL; (* X distance trigger *)
    gptYDelta: CARDINAL; (* Y distance trigger *)
END;
```

CONST

```
    GPAllocated = -1; (* allocated by another user *)
    GPNoController = 0;
    GPMouse = 1;
    GPRelJoystick = 2;
    GPAbsJoystick = 2;
```

END GamePortDevice.

DEFINITION MODULE Gels;

(** -----

Commodore Amiga Gels (graphic elements) module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 09-Jan-86

Version : 0.00a 09-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, WORD, ADDRESS;

FROM Rasters IMPORT RastPort;

FROM Views IMPORT ViewPort;

TYPE

VSpriteFlags = (vsprite, (* set if VSprite, clear if Bob *)
SaveBack, (* set if background is to be saved/restored *)
Overlay, (* set to mask image of Bob onto background *)
MustDraw, (* set if VSprite must be drawn *)
VSF4, VSF5, VSF6, VSF7,
BackSaved, (* this Bob's background has been saved *)
BobUpdate, (* temporary flag *)
GelGone, (* set if gel is completely clipped (offscreen) *)
VSOoverflow, (* VSprite overflow *)
VSF12, VSF13, VSF14, VSF15);
VSpriteFlagSet = SET OF VSpriteFlags;

TYPE

BobFlags = (SaveBob, (* set to not erase Bob *)
BobIsComp, (* set to identify Bob as AnimComp *)
BFU2, BFU3, BFU4, BFU5, BFU6, BFU7,
BWaiting, (* set while Bob is waiting on 'after' *)
BDrawn, (* set when Bob is drawn this DrawG pass *)

```

    BobsAway, (* set to initiate removal of Bob *)
    BobInx, (* set when Bob is completely removed *)
    SavePreserve, (* for back-restore during double-buffer *)
    OutStep, (* for double-clearing if double-buffer *)
    BFU14,BFU15);

```

```

BobFlagSet = SET OF BobFlags;

```

TYPE

```

VUserStuff = CARDINAL;
BUserStuff = CARDINAL;
AUserStuff = CARDINAL;

```

TYPE

```

BobPtr = POINTER TO Bob;
AnimObPtr = POINTER TO AnimOb;
VSpritePtr = POINTER TO VSprite;
AnimCompPtr = POINTER TO AnimComp;
DBufPacketPtr = POINTER TO DBufPacket;

```

TYPE

```

VSprite = RECORD
    NextVSprite: VSpritePtr;
    PrevVSprite: VSpritePtr;
    DrawPath: VSpritePtr;
    ClearPath: VSpritePtr;
    OldY: CARDINAL;
    OldX: CARDINAL;
    Flags: VSpriteFlagSet;
    Y: CARDINAL; (* screen position *)
    X: CARDINAL;
    Height: CARDINAL;
    Width: CARDINAL; (* number of words per row of image data *)
    Depth: CARDINAL; (* number of planes of data *)
    MeMask: BITSET; (* which types can collide with this sprite *)
    HitMask: BITSET; (* which types this sprite can collide with *)
    ImageData: ADDRESS;
    BorderLine: ADDRESS; (* logical OR of all VSprite bits *)
    CollMask: ADDRESS;
    SprColors: ADDRESS; (* sprite colours *)
    VSBob: BobPtr;

```

```

    PlanePick: BYTE;
    PlaneOnOff: BYTE;
    VUserExt: VUserStuff;
END;

```

TYPE

Bob = RECORD

```

    Flags: BobFlagSet;
    SaveBuffer: ADDRESS; (* buffer pointer for background save *)
    ImageShadow: ADDRESS;
    Before: BobPtr; (* draw this Bob before Bob pointed to by before *)
    After: BobPtr; (* draw this Bob after Bob pointed to by after *)
    BobVSprite: VSpritePtr; (* this Bob's VSprite definition *)
    BobComp: AnimCompPtr; (* pointer to this Bob's AnimComp def *)
    DBuffer: DBufPacketPtr; (* pointer to this Bob's dBuf packet *)
    BUserExt: BUserStuff; (* Bob user extension *)
END;

```

TYPE

AnimComp = RECORD

```

    Flags: BITSET;
    Timer: INTEGER;
    TimeSet: INTEGER;
    NextComp: AnimCompPtr;
    PrevComp: AnimCompPtr;
    NextSeq: AnimCompPtr;
    PrevSeq: AnimCompPtr;
    AnimCRoutine: PROCEDURE;
    YTrans: INTEGER; (* initial y translation *)
    XTrans: INTEGER; (* initial x translation *)
    HeadOb: AnimObPtr;
    AnimBob: BobPtr;
END;

```

TYPE

AnimOb = RECORD

```

    NextOb: AnimObPtr;
    PrevOb: AnimObPtr;
    Clock: LONGINT;
    AnOldY: INTEGER; (* old y,x coordinates *)
    AnOldX: INTEGER;
    AnY: INTEGER; (* y,x coordinates of the AnimOb *)

```

```

    AnX: INTEGER;
    YVel: INTEGER; (* velocities of this object *)
    XVel: INTEGER;
    YAccel: INTEGER; (* accelerations of this object *)
    XAccel: INTEGER;
    RingYTrans: INTEGER; (* ring translation values *)
    RingXTrans: INTEGER;
    AnimORoutine: PROCEDURE;
    HeadComp: AnimCompPtr; (* pointer to first component *)
    AUserExt: AUserStuff; (* AnimOb user extension *)
END;

```

TYPE

```

    DBufPacket = RECORD
        BufY: INTEGER; (* save the other buffers coordinates *)
        BufX: INTEGER;
        BufPath: VSpritePtr; (* carry the draw path over the gap *)
        BufBuffer: ADDRESS;
    END;

```

TYPE

```

    collTablePtr = POINTER TO collTable;
    collTable = ARRAY [0..15] OF PROCEDURE;

```

TYPE

```

    CARDINAL8 = ARRAY [0..7] OF CARDINAL;
    ADDRESS8 = ARRAY [0..8] OF ADDRESS;

```

TYPE

```

    GelsInfoPtr = POINTER TO GelsInfo;
    GelsInfo = RECORD
        sprRsrvd: BYTE;
        flags: BYTE; (* system use *)
        gelHead: VSpritePtr; (* dummy vSprites for list management *)
        gelTail: VSpritePtr;
        nextLine: POINTER TO CARDINAL8; (* sprite available lines *)
        lastColour: POINTER TO ADDRESS8; (* color last assigned *)
        collHandler: collTablePtr;
        leftmost: INTEGER;
        rightmost: INTEGER;
        topmost: INTEGER;
        bottommost: INTEGER;
    END;

```

```

firstBlissObj: ADDRESS; (* system use *)
lastBlissObj: ADDRESS;
END;

```

(* These bit descriptors are used by the GEL collide routines. These bits are set in the hitMask and meMask variables of a GEL to describe whether or not these types of collisions can affect the GEL. BoundaryHit is described further below; this bit is permanently assigned as the boundary-hit flag. The other bit GELHit is meant only as a default to cover any GEL hitting any other; the user may redefine this bit. *)

```

CONST
  BorderHit = 0;

```

(* These bit descriptors are used by the GEL boundry hit routines. When the user's boundry-hit routine is called (via the argument set by a call to SetCollision) the first argument passed to the user's routine is the address of the GEL involved in the boundry-hit, and the second argument has the appropriate bits set to describe which boundry was surpassed. *)

```

CONST
  TopHit = 1;
  BottomHit = 2;
  LeftHit = 4;
  RightHit = 8;

```

```

PROCEDURE AddAnimOb(VAR anOb: AnimOb; anKey: ADDRESS; VAR rport: RastPort);
(* add an AnimOb to the linked list of AnimObs.

```

```

    anOb: the AnimOb to add to the list.
    anKey: address of the pointer to the first AnimOb in the list,
           0 => none.
    rport: RastPort to add AnimOb to. *)

```

```

PROCEDURE AddBob(VAR bob: Bob; VAR rport: RastPort);
(* add a Bob to the current gel list.

```

```

    bob: the bob to be added to the gel list.
    rport: RastPort to add bob to. *)

```

```

PROCEDURE AddVSprite(VAR vs: VSprite; VAR rport: RastPort);

```

(* add a VSprite to the current gel list.

vs: the VSprite to add to the gel list.
rport: RastPort to add VSprite to. *)

PROCEDURE Animate(key: ADDRESS; VAR rport: RastPort);
(* process every AnimOb in the current animation list.

key: the address of the variable that points to the head AnimOb.
rport: pointer to a RastPort. *)

PROCEDURE DoCollision(VAR rport: RastPort);
(* tests every gel in gel list for collision.

rport: the rport to test collisions in. *)

PROCEDURE DrawGList(VAR rport: RastPort; VAR vport: ViewPort);
(* process the gel list, queueing VSprites, drawing Bobs.

rport: RastPort where Bobs will be drawn.
vport: the view port for beam synchronising. *)

PROCEDURE GetGBuffers(VAR anOb: AnimOb; VAR rport: RastPort;
db: BOOLEAN): BOOLEAN;

(* attempts to allocate all the buffers of an AnimOb.

anOb: the AnimOb to allocate buffers for.
rport: the current raster port.
db: double buffer indicator, TRUE => double buffering required.

returns: TRUE => buffers allocated OK, otherwise not enough memory. *)

PROCEDURE InitGels(VAR head, tail: VSprite; VAR gInfo: GelsInfo);
(* initialises a gel list - must be called before using gels.

head: the VSprite to be used as the head of the list.
tail: the VSprite to be used as the tail of the list.
gInfo: the GelsInfo structure to be initialised. *)

PROCEDURE InitGMasks(VAR anOb: AnimOb);
(* initialise all the masks of an AnimOb.

anOb: the AnimOb to initialise the masks of. *)

PROCEDURE InitMasks(VAR vs: VSprite);

(* initialise the BorderLine and CollMask masks of a VSprite.

vs: the VSprite to initialise the masks of. *)

PROCEDURE RemIBob(VAR bob: Bob; VAR rport: RastPort; VAR vp: ViewPort);

(* immediately remove a Bob from the gel list and raster port.

bob: the Bob to be removed.

rport: the rasterport if the Bob is to be erased.

vport: the view port for beam synchronising. *)

PROCEDURE RemVSprite(VAR vs: VSprite);

(* remove a VSprite from the current gel list.

vs: the VSprite to be removed from the gel list. *)

PROCEDURE SetCollision(num: LONGCARD; routine: PROC; VAR gInfo: GelsInfo);

(* set a pointer to a user collision routine.

num: the collision vector number.

routine: the user collision routine.

gInfo: the GelsInfo structure in which to set the vector. *)

PROCEDURE SortGList(VAR rport: RastPort);

(* sort the current gel list according to the y,x coordinates.

rp: the RastPort structure containing the GelsInfo to sort. *)

END Gels.

DEFINITION MODULE GraphicsBase;

(** -----

Commodore Amiga graphics library data module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 28-Jan-86

Version : 0.00a 28-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM Lists IMPORT List;
FROM Text IMPORT TextFontPtr;
FROM Tasks IMPORT TaskPtr;
FROM Libraries IMPORT Library;
FROM Views IMPORT ViewPtr;
FROM Copper IMPORT copinitptr;
FROM Blitter IMPORT bltnodeptr;
FROM Interrupts IMPORT Interrupt;

TYPE

GfxBasePtr = POINTER TO RECORD

LibNode: Library;
ActiView: ViewPtr;
copInit: copinitptr; (* ptr to copper start up list *)
cia: ADDRESS; (* for 8520 resource use *)
blitter: ADDRESS; (* for future blitter resource use *)
LOFlist: ADDRESS;
SHFlist: ADDRESS;
blthd: bltnodeptr;
blttl: bltnodeptr;
bsblthd: bltnodeptr;

```

bsblttl: bltnodeptr;
vbsrv: Interrupt;
timsrv: Interrupt;
bltsrv: Interrupt;
TextFonts: List;
DefaultFont: TextFontPtr;
Modes: BITSET; (* copy of current first bplcon0 *)
VBlank: BYTE;
Debug: BYTE;
BeamSync: INTEGER;
systembplcon0: INTEGER;
SpriteReserved: BYTE;
bytereserved: BYTE;
Flags: BITSET;
BlitLock: CARDINAL;
BlitNest: CARDINAL;

BlitWaitQ: List;
BlitOwner: TaskPtr;
TOFWaitQ: List;
reserved: ARRAY [0..1] OF LONGCARD;
END;

```

```

END GraphicsBase.

```

DEFINITION MODULE GraphicsLibrary;

(** -----

Commodore Amiga graphics basetype module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 19-Dec-85

Version : 0.80a 19-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;

CONST

GraphicsName = "graphics.library";

VAR

GraphicsBase: ADDRESS;

TYPE

DrawingModes = (Jam2, (* Jam two colours into raster *)
Complement, (* XOR bits into raster *)
InverseVid); (* inverse video for drawing modes *)
DrawingModeSet = SET OF DrawingModes;

CONST

Jam1 = DrawingModeSet{}; (* Jam one colour into raster *)

TYPE

RectanglePtr = POINTER TO Rectangle;
Rectangle = RECORD
MinX: INTEGER;
MinY: INTEGER;

```

        MaxX: INTEGER;
        MaxY: INTEGER;
    END;

```

```

TYPE

```

```

    PlanePtr = ADDRESS;

```

```

TYPE

```

```

    BitMapPtr = POINTER TO BitMap;

```

```

    BitMap = RECORD

```

```

        BytesPerRow: CARDINAL;

```

```

        Rows: CARDINAL;

```

```

        Flags: BYTE;

```

```

        Depth: BYTE;

```

```

        Pad: CARDINAL;

```

```

        Planes: ARRAY [0..7] OF PlanePtr;

```

```

    END;

```

```

PROCEDURE BitClear(memBlk: ADDRESS; bytecount: LONGCARD; flags: BITSET);
    (* clear a block of memory words to zero.

```

```

    memBlk: the start address of the memory to be cleared.

```

```

    bytecount: the number of bytes to clear.

```

```

    flags: {0} => wait for blit to finish.

```

```

           {1} => row/bytes per row mode for bytecount. *)

```

```

PROCEDURE InitBitMap(VAR bm: BitMap; d,w,h: CARDINAL);

```

```

    (* initialise bitmap structure with given values.

```

```

    bm: the bitmap structure to initialise.

```

```

    d: depth of bitmap.

```

```

    w: width of bitmap.

```

```

    h: height of bitmap. *)

```

```

END GraphicsLibrary.

```

DEFINITION MODULE IconLibrary;

(** -----

Commodore Amiga icon library module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 28-Jan-86

Version : 0.00a 28-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;

FROM Workbench IMPORT WBOBJECTPtr, WBOBJECT, FreeList, DiskObject;

CONST

IconName = "icon.library";

VAR

IconBase: ADDRESS;

PROCEDURE AddFreeList(VAR free: FreeList; mem: ADDRESS; len: LONGCARD):INTEGER;

(* add memory to the free list.

free: the freelist structure.

mem: the base of the memory to be recorded.

len: the length of the memory to be recorded.

returns: 0 => call failed, otherwise OK. *)

PROCEDURE AllocWBOBJECT(): WBOBJECTPtr;

(* allocate a Workbench object.

returns: 0 => call failed to allocate an object, otherwise pointer
to the new object. *)

PROCEDURE FreeFreeList(VAR free: FreeList);

(* free all the memory in the free list,

free: the free list to free. *)

PROCEDURE FreeWBOobject(VAR obj: WBOobject);

(* free all memory for a Workbench object,

obj: the object to free. *)

PROCEDURE GetIcon(VAR name: ARRAY OF CHAR; VAR icon: DiskObject;

VAR free: FreeList): INTEGER;

(* read in a DiskObject structure from disk,

name: the name of the icon to read,

icon: the DiskObject to read into,

free: the free list to append memory onto,

returns: 0 => call failed, otherwise read in OK. *)

PROCEDURE GetWBOobject(VAR name: ARRAY OF CHAR): WBOobjectPtr;

(* read in a Workbench object,

name: the name of the object,

returns: 0 => call failed, otherwise a pointer to the Workbench
object. *)

PROCEDURE PutIcon(VAR name: ARRAY OF CHAR; VAR icon: DiskObject): INTEGER;

(* write out a DiskObject structure to disk,

name: the name of the icon,

icon: the DiskObject to write,

returns: 0 => call failed, otherwise object written OK. *)

PROCEDURE PutWBOobject(VAR name: ARRAY OF CHAR; VAR object: WBOobject): INTEGER;

(* write a Workbench object to disk,

name: the name of the Workbench object,

object: the Workbench object to write. *)

END IconLibrary.

DEFINITION MODULE InOut;

(** -----

Commodore Amiga standard InOut module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 03-Feb-86

Version : 0.00a 03-Feb-86 Paul Curtis, TDI.
Original according to "Programming in
Modula-2" (third edition.)

*)

CONST EOL = 12C; (*LF*)

VAR Done: BOOLEAN;

termCH: CHAR; (*terminating character in ReadInt, ReadCard*)

PROCEDURE OpenInput(VAR defext: ARRAY OF CHAR);

(*request a file name and open input file "in".

Done := "file was successfully opened".

If TRUE, subsequent input is read from this file.

If name ends with ".", append extension defext*)

PROCEDURE OpenInputFile(VAR FileName: ARRAY OF CHAR);

(* as above by passing filename down as a parameter *)

PROCEDURE OpenOutput(VAR defext: ARRAY OF CHAR);

(*request a file name and open output file "out"

Done := "file was successfully opened.

If TRUE, subsequent output is written on this file*)

PROCEDURE OpenOutputFile(VAR FileName: ARRAY OF CHAR);

(* as above by passing filename down as a parameter *)

PROCEDURE CloseInput;

(*closes input file; returns input to terminal*)

```

PROCEDURE CloseOutput;
  (*closes output file; returns output to terminal*)

PROCEDURE Read(VAR ch: CHAR);
  (* Done := NOT in.eof
   ie Done := TRUE if Read was succesful and
   Done := FALSE when the Read returns the
   end of file character in the file "in" *)

PROCEDURE ReadString(VAR s: ARRAY OF CHAR);
  (*read string, i.e. sequence of characters *)

PROCEDURE ReadInt(VAR x: INTEGER);
  (*read string and convert to integer. Syntax:
   integer = ["+"|"~"] digit {digit}.
   Leading blanks are ignored.
   Done := "integer was read"*)

PROCEDURE ReadCard(VAR x: CARDINAL);
  (*read string and convert to cardinal. Syntax:
   cardinal = digit {digit}.
   Leading blanks are ignored.
   Done := "cardinal was read"*)

PROCEDURE Write(ch: CHAR);
  (* Writes any 8 bit character out *)

PROCEDURE WriteLn;
  (* terminate line by writing ASCII.CR and ASCII.LF *)

PROCEDURE WriteString ( VAR s : ARRAY OF CHAR );
  (* the string is a sequence of characters ening with ASCII.NUL *)

PROCEDURE WriteInt(x: INTEGER; n: CARDINAL);
  (*write integer x with (at least) n characters on file "out".
   If n is greater than the number of digits needed,
   blanks are added preceding the number*)

PROCEDURE WriteCard(x, n: CARDINAL);
PROCEDURE WriteOct(x, n: CARDINAL);
PROCEDURE WriteHex(x, n: CARDINAL);

END InOut.
```

DEFINITION MODULE InputDevice;

(** -----

Commodore Amiga input device module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 29-Jan-86

Version : 0.00a 29-Jan-86 Paul Curtis, TDI.
Original

*)

FROM IO IMPORT CmdNonStd;

CONST

InputDeviceName = "input.device";

CONST

INDAddHandler = CmdNonStd + 0;

INDRemHandler = CmdNonStd + 1;

INDWriteEvent = CmdNonStd + 2;

INDSetThresh = CmdNonStd + 3;

INDSetPeriod = CmdNonStd + 4;

END InputDevice.

DEFINITION MODULE InputEvents;

(** -----
Commodore Amiga input event module
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved
----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 30-Jan-86

Version : 0.00a 30-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM TimerDevice IMPORT TimeVal;

TYPE

IEClass = (null, (* a NOP input event *)
rawKey, (* a raw keycode from the keyboard device *)
rawMouse, (* the raw mouse report from the game port device *)
event, (* a private console event *)
pointerPos, (* a pointer position report *)
timer, (* a timer event *)
gadgetDown, (* select down over Gadget *)
gadgetUp, (* select released over same Gadget *)
requester, (* some requester activity has taken place *)
menuList, (* this is a menu number transmission *)
closeWindow, (* user selected active window's close gadget *)
sizeWindow, (* this window has a new size *)
refreshWindow, (* a window needs to be refreshed *)
newPrefs, (* new preferences are available *)
diskRemoved, (* the disk has been removed *)
diskInserted, (* the disk has been inserted *)
activeWindow, (* the window is about to be been made active *)
inactiveWindow); (* the window is about to be made inactive *)

CONST

(* RAWKEY *)

UpPrefix = 80H; (* key was released if set *)

KeyCodeFirst = 00H; (* first keystroke code *)

KeyCodeLast = 77H; (* last keystroke code *)

CommCodeFirst = 78H; (* first control code from 6500/1 *)

CommCodeLast = 7FH; (* last control code from 6500/1 *)

(* ANSI *)

COFirst = 000H;

COLast = 01FH;

ASCIIFirst = 020H;

ASCIILast = 07EH;

ASCIIDEL = 07FH;

CIFirst = 080H;

CILast = 09FH;

LatinIFirst = 0A0H;

LatinILast = 0FFH;

(* RAWMOUSE - may be modified by UpPrefix *)

LButton = 068H; (* left button down *)

RButton = 069H; (* right button down *)

MButton = 06AH; (* middle button down *)

NoButton = 0FFH;

(* EVENT *)

NewActive = 1; (* active input window changed *)

(* REQUESTER *)

ReqSet = 1; (* first requester opened in window *)

ReqClear = 0; (* last requester closed in window *)

TYPE

IEQualifier = (LeftShift,
RShift,
CapsLock,
Control,
LeftAlt,
RightAlt,
LeftCommand,
RightCommand,

```
NumericPad,  
Repeat,  
Interrupt,  
MultiBroadcast,  
LeftButton,  
RightButton,  
MiddleButton,  
RelativeMouse);
```

TYPE

```
IEQualifierSet = SET OF IEQualifier;
```

TYPE

```
InputEventPtr = POINTER TO InputEvent;
```

```
InputEvent = RECORD
```

```
    ieNextEvent: InputEventPtr; (* chronologically next evnt *)
```

```
    ieClass: IEClass; (* the input event class *)
```

```
    ieSubClass: BYTE; (* optional subclass of the class *)
```

```
    ieCode: CARDINAL; (* the input event code *)
```

```
    ieQualifier: IEQualifierSet; (* qualifiers for the event *)
```

```
    CASE BOOLEAN OF
```

```
        FALSE:
```

```
            ieX: INTEGER;
```

```
            ieY: INTEGER; |
```

```
        TRUE:
```

```
            ieAddr: ADDRESS;
```

```
    END;
```

```
    ieTimeStamp: TimeVal;
```

```
END;
```

END InputEvents.

DEFINITION MODULE IntBits;

(** -----

Commodore Amiga interrupt bits module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 29-Jan-86

Version : 0.00a 29-Jan-86 Paul Curtis, TDI.
Original

*)

(* bits in the interrupt request and enable registers *)

TYPE

IntBit = (TBE, (* serial port transmit buffer empty *)
DskBlk, (* disk block done *)
SoftInt, (* software interrupt request *)
Ports, (* I/O ports and timers *)
Coper, (* coprocessor *)
VertBlank, (* start of vertical blank *)
EndBlit, (* Blitter finished *)
Aud0, (* audio channel 0 block finished *)
Aud1, (* audio channel 1 block finished *)
Aud2, (* audio channel 2 block finished *)
Aud3, (* audio channel 3 block finished *)
RBF, (* serial port receive buffer full *)
DskSync, (* Disk resynchronized *)
External, (* external interrupt *)
IntEn, (* Master interrupt, enable only *)
IntSetClr); (* standard set/clear bit *)

TYPE

IntBitSet = SET OF IntBit;

END IntBits.

DEFINITION MODULE Interrupts;

(** -----

Commodore Amiga interrupts module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 04-Dec-85

Version : 0.00a 04-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;
FROM Nodes IMPORT Node, NodePtr;
FROM Lists IMPORT List;

TYPE

InterruptPtr = POINTER TO Interrupt;
Interrupt = RECORD
 isNode: Node;
 isData: ADDRESS; (* server data segment *)
 isCode: PROCEDURE; (* server code entry *)
END;

(* These are for exec use only, not very useful. *)

TYPE

IntVectorPtr = POINTER TO IntVector;
IntVector = RECORD
 ivData: ADDRESS;
 ivCode: PROCEDURE;
 ivNode: NodePtr;
END;

SoftIntList = RECORD
 shList: List;


```
shPad: CARDINAL;  
END;
```

```
CONST
```

```
SIPriMask = 0F0H;
```

```
PROCEDURE AddIntServer(intNum: CARDINAL; int: InterruptPtr);  
(* add an interrupt server to the system.
```

```
intNum: the Portia interrupt bit to add this server to.  
int: pointer to an interrupt server node. *)
```

```
PROCEDURE Cause(int: IntVectorPtr);  
(* cause a software interrupt.
```

```
int: pointer to the interrupt node. *)
```

```
PROCEDURE Disable;  
(* disable interrupts. *)
```

```
PROCEDURE Enable;  
(* enable interrupts. *)
```

```
PROCEDURE Forbid;  
(* forbids task switching. *)
```

```
PROCEDURE Permit;  
(* permits task switching. *)
```

```
PROCEDURE RemIntServer(intNum: CARDINAL; int: InterruptPtr);  
(* remove an interrupt server.
```

```
intNum: the Portia interrupt bit to remove this server from.  
int: pointer to an interrupt server node. *)
```

```
PROCEDURE SetIntVector(intNum: CARDINAL; int: IntVectorPtr): IntVectorPtr;  
(* set a system interrupt vector.
```

```
intNum: the Portia interrupt bit number to set vector of.
```

int: the new interrupt handler.

returns: the old interrupt handler. *)

PROCEDURE SetsR(newSR: BITSET; mask: BITSET): BITSET;

(* get and/or set processor status register.

newSR: the new value for the status register, according to mask.

mask: the bits to change in the status register.

returns: the old status register value. *)

PROCEDURE SuperState(): ADDRESS;

(* enter supervisor state with user stack.

returns: system stack pointer. *)

PROCEDURE UserState(sysStack: ADDRESS);

(* return to user state with user stack.

sysStack: the value returned by SuperState. *)

END Interrupts.

This page blank

DEFINITION MODULE Intuition ;

(% -----
 TDI Modula-2/Amiga : Intuition
----- %)

(% ----- %)
(% (c) Copyright TDI Software Ltd. 1986. All Rights Reserved %)
(% ----- %)

FROM SYSTEM IMPORT ADDRESS, BYTE, LONGWORD ;
FROM Layers IMPORT LayerInfo, LayerPtr, ClipRect ;
FROM Ports IMPORT Message, MsgPortPtr, MsgPort ;
FROM GraphicsLibrary IMPORT DrawingModes, BitMapPtr, BitMap ;
FROM Rasters IMPORT RastPort, RastPortPtr ;
FROM Text IMPORT TextAttrPtr ;
FROM Views IMPORT ViewPort ;

(% NOTE :

Pointer name types commented by '(**)' and '(**)' denote the true type of the pointer. They have not been defined with that type to cut down inter-definition module references. This will not cause any problems as the type ADDRESS is compatible with all pointer types.

Some of the structure definitions are grouped in Intuition.def rather than the specialist module for that type. This is because their inclusion in the specialist modules would cause cyclic imports between the definition modules (eg. module Windows needs the structure Requester, and module Requesters needs the structure Window!).

*)

(% Intuition library control %)

CONST

IntuitionName = "intuition.library" ;

VAR

IntuitionBase : ADDRESS ; (% library base address, preset to 0 for %)

(* unloaded library *)

(* PropInfo *)

(* This is the special data required by the proportional Gadget typically,
this data will be pointed to by the Gadget variable SpecialInfo. *)

TYPE

```
PropFlags = (  
    AutoKnob,      (* Give me auto-knob *)  
    FreeHoriz,     (* the knob can move horizontally *)  
    FreeVert,      (* the knob can move vertically *)  
    PropBorderless, (* if set, no border will be rendered *)  
    KnobHit,       (* set when this Knob is hit *)  
    Pr5, Pr6, Pr7, Pr8 (* resrvd *)  
);
```

PropFlagsSet = SET OF PropFlags ;

PropInfo = RECORD

```
    Flags : PropFlagsSet ; (* general purpose flag bits *)  
    HorizPot : CARDINAL ; (* FixedPoint horizontal quantity % *)  
    VertPot : CARDINAL ; (* FixedPoint vertical quantity % *)  
    HorizBody : CARDINAL ; (* horizontal Body *)  
    VertBody : CARDINAL ; (* vertical Body *)  
    CWidth : CARDINAL ; (* Container width and countainer height *)  
    CHeight : CARDINAL ; (* (with any relativity absolved) *)  
    HPotRes, VPotRes : CARDINAL ; (* pot increments *)  
    LeftBorder : CARDINAL ; (* Container borders *)  
    TopBorder : CARDINAL ; (* Container borders *)  
END ; (* OF RECORD *)
```

(* Notes :

HorizPot, VertPot: You initialize the Pot variables before the Gadget is added to the system. Then you can look at them for the current settings any time, even while User is playing with this Gadget. To adjust these after the Gadget is added to the System, use ModifyProp(); The Pots are the actual proportional settings, where a value of zero means zero and a value of MaxPot means that the Gadget is set to its maximum setting.

HorizBody, VertBody: The FixedPoint Body variables describe what percentage

of the entire body of stuff referred to by this Gadget is actually shown at one time. This is used with the AutoKnob routines, to adjust the size of the AutoKnob according to how much of the data can be seen. This is also used to decide how far to advance the Pots when User hits the Container of the Gadget. For instance, if you were controlling the display of a 5-line Window of text with this Gadget, and there was a total of 15 lines that could be displayed, you would set the VertBody value to

$(\text{MaxBody} / (\text{TotalLines} / \text{DisplayLines})) = \text{MaxBody} / 3.$

Therefore, the AutoKnob would fill 1/3 of the container, and if User hits the Container outside of the knob, the pot would advance 1/3 (plus or minus). If there's no body to show, or the total amount of displayable info is less than the display area, set the Body variables to the MAX. To adjust these after the Gadget is added to the System, use ModifyProp();

*)

CONST

```
KnobHMin =      6 ;      (* minimum horizontal size of the Knob *)
KnobVMin =      4 ;      (* minimum vertical size of the Knob *)
MaxBody = 0FFFFH ;      (* maximum body value *)
MaxXPot = 0FFFFH ;      (* maximum pot value *)
```

(* StringInfo *)

TYPE

(* This is the special data required by the string Gadget typically, this data will be pointed to by the Gadget variable SpecialInfo. *)

StringInfo = RECORD

(* you initialize these variables, and then Intuition maintains them *)

Buffer : ADDRESS ; (* buffer containing start and final string*)

UndoBuffer : ADDRESS ; (* optional buffer for current entry *)

BufferPos : INTEGER ; (* character position in Buffer *)

MaxChars : INTEGER ; (* max num chars in Buffer (incl NULL) *)

DispPos : INTEGER ; (* Buffer position of first disp char *)

(* Intuition initializes and maintains these variables for you *)

UndoPos : INTEGER ; (* char position in the undo buffer *)

NumChars : INTEGER ; (* num of chars currently in Buffer *)

DispCount : INTEGER ; (* num whole chars visible in Container *)

CLeft, CTop : INTEGER; (* topleft offset of the container *)

Layer : ADDRESS (**LayerPtr**); (* RastPort containing Gadget *)

```

    LongInt : LONGINT ;
    AltKeyMap : ADDRESS (**KeyMapPtr**) ;
END ; (* OF RECORD *)

```

(* Notes:

AltKeyMap: If you want this Gadget to use your own Console keymapping, you set the AltKeyMap bit in the Activation flags of the Gadget, and then set this variable to point to your keymap. If you don't set the AltKeyMap, you'll get the standard ASCII keymapping.

*)

(* Intuition Text *)

(* IntuitionText is a series of strings that start with a screen location (always relative to the upper-left corner of something) and then the text of the string. The text is null-terminated. *)

TYPE

IntuitionTextPtr = POINTER TO IntuitionText ;

IntuitionText = RECORD

```

    FrontPen,
    BackPen : BYTE ;      (* the pen numbers for the rendering *)
    DrawMode : BYTE ;     (* the mode for rendering the text *)
    LeftEdge : INTEGER ;  (* relative start location *)
    TopEdge : INTEGER ;   (* relative start location *)
    ITextFont : TextAttrPtr ; (* NULL=default font*)
    IText : ADDRESS ;      (* pointer to null-terminated text *)
    NextText : IntuitionTextPtr ;
                        (* continuation to TxWrite more text *)

```

END ; (* OF RECORD *)

(* Border *)

(* Data type Border, used for drawing a series of lines which is intended for use as a border drawing, but which may, in fact, be used to render any arbitrary vector shape. The routine DrawBorder sets up the RastPort with the appropriate variables, then does a Move to the first coordinate, then

does Draws to the subsequent coordinates. After all the Draws are done, if NextBorder is non-zero we call DrawBorder recursively.

*)

BorderPtr = POINTER TO Border ;

Border = RECORD

LeftEdge, TopEdge : INTEGER ; (* initial offsets from the origin *)
FrontPen, BackPen : BYTE ; (* pens numbers for rendering *)
DrawMode : BYTE ; (* mode for rendering *)
Count : BYTE ; (* number of XY pairs *)
XY : ADDRESS ; (* vector coord pairs rel to LeftTop*)
NextBorder : BorderPtr ; (* pointer to any other Border too *)
END ; (* OF RECORD *)

(* Image *)

(* This is a brief image structure for very simple transfers of image data to a RastPort. *)

ImagePtr = POINTER TO Image ;

Image = RECORD

LeftEdge : INTEGER ; (* starting offset relative to some origin *)
TopEdge : INTEGER ; (* starting offsets relative to some origin *)
Width : INTEGER ; (* pixel size (though data is word-aligned) *)
Height, Depth : INTEGER ; (* pixel sizes *)
ImageData : ADDRESS ; (* pointer to the actual word-aligned bits *)
PlanePick, PlaneOnOff : BYTE ; (* see notes below *)
NextImage : ImagePtr ;
END ; (* OF RECORD *)

(* Notes:

The PlanePick and PlaneOnOff variables work much the same way as the equivalent GELS Bob variables. It's a space-saving mechanism for image data. Rather than defining the image data for every plane of the RastPort, you need define data only for the planes that are not entirely zero or one. As you define your Imagery, you will often find that most of the planes ARE just as color selectors. For instance, if you're designing a two-color Gadget to use colors two and three, and the Gadget will reside in a five-plane display, bit plane zero of your imagery would be all ones, bit plane one would have data that describes the imagery, and bit planes two

through four would be all zeroes. Using these flags allows you to avoid wasting all that memory in this way: first, you specify which planes you want your data to appear in using the PlanePick variable. For each bit set in the variable, the next "plane" of your image data is blitted to the display. For each bit clear in this variable, the corresponding bit in PlaneOnOff is examined. If that bit is clear, a "plane" of zeroes will be used. If the bit is set, ones will go out instead. So, for our example:

```
Gadget.PlanePick = 002H
```

```
Gadget.PlaneOnOff = 001H
```

Note that this also allows for generic Gadgets, like the System Gadgets, which will work in any number of bit planes. Note also that if you want an Image that is only a filled rectangle, you can get this by setting PlanePick to zero (pick no planes of data) and set PlaneOnOff to describe the pen color of the rectangle.

If the NextImage variable is not NULL, Intuition presumes that it points to another Image structure with another Image to be rendered.

*)

(* IntuiMessage *)

```
IDCMPFlags = (
    SizeVerify,
    NewSize,
    RefreshWindow,
    MouseButtons,
    MouseMove,
    GadgetDown,
    GadgetUp,
    ReqSet,
    MenuPick,
    CloseWindow,
    RawKey,
    ReqVerify,
    ReqClear,
    MenuVerify,
    NewPrefs,
    DiskInserted,
    DiskRemoved,
    WBenchMessage,
    ActiveWindow,
```

```

InactiveWindow,
DeltaMove,
ID21, ID22, ID23, ID24, ID25, ID26, ID27, ID28, ID29, ID30,
(* reserved *)
LonelyMessage
) ;

```

```
IDCMPFlagsSet = SET OF IDCMPFlags ;
```

(* Notes:

LonelyMessage: The IDCMP Flags do not use this special bit, which is cleared when Intuition sends its special message to the Task, and set when Intuition gets its Message back from the Task. Therefore, I can check here to find out fast whether or not this Message is available for me to send.

*)

```
IntuiMessagePtr = POINTER TO IntuiMessage ;
```

```
IntuiMessage = RECORD
```

```

    ExecMessage : Message;
    Class : IDCMPFlagsSet ;
    Code : CARDINAL ;
    Qualifier : CARDINAL ;
    IAddress : ADDRESS ;
    MouseX, MouseY : INTEGER ;
    Seconds, Micros : LONGCARD ;
    IDCMPWindow : ADDRESS (**WindowPtr**) ;
    SpecialLink : IntuiMessagePtr;
END ; (* OF RECORD *)

```

(* Notes :

The Class bits correspond directly with the IDCMP Flags, except for the special bit LonelyMessage.

The Code field is for special values like Menu number.

The Qualifier field is a copy of the current InputEvent's Qualifier.

IAddress contains particular addresses for Intuition functions, like the pointer to the Gadget or the Screen.

When getting mouse movement reports, any event you get will have the the

mouse coordinates in these variables. The coordinates are relative to the upper-left corner of your Window (GimmeZeroZero notwithstanding).

The time values are copies of the current system clock time. Micros are in units of microseconds, Seconds in seconds.

The IDCMPWindow variable will always have the address of the Window of this IDCMP.

*)

(* IDCMP Codes *)

CONST

(* This group of codes is for the MenuVerify function *)

MenuHot = 00001H ; (* IntuiWants verification or MenuCANCEL *)

MenuCancel = 00002H ; (* HOT Reply of this cancels Menu operation *)

MenuWaiting = 00003H ; (* Intuition simply wants a ReplyMsg() ASAP *)

(* This group of codes is for the WBENCHMESSAGE messages *)

WBenchOpen = 00001H ;

WBenchClose = 00002H ;

(* Remember *)

(* This structure is used for remembering what memory has been allocated to date by a given routine, so that a premature abort or systematic exit can deallocate memory cleanly, easily, and completely. *)

TYPE

RememberPtr = POINTER TO Remember ;

Remember = RECORD

NextRemember : RememberPtr ;

RememberSize : LONGCARD ;

Memory : ADDRESS ;

END ; (* OF RECORD *)

(* ----- *)

(* Various pointers for inter structure stuff *)

TYPE

```
WindowPtr = POINTER TO Window ;
RequesterPtr = POINTER TO Requester ;
MenuPtr = POINTER TO Menu ;
ScreenPtr = POINTER TO Screen ;
GadgetPtr = POINTER TO Gadget ;
MenuItemPtr = POINTER TO MenuItem ;
```

(* Windows *)

TYPE

```
WindowFlags = (
    (* Flags requested (not set directly) by the application *)
    WindowSizing,    (* include sizing system-gadget? *)
    WindowDrag,      (* include dragging system-gadget? *)
    WindowDepth,     (* include depth arrangement gadget? *)
    WindowClose,     (* include close-box system-gadget? *)
    SizeBRight,      (* size gadget uses right border *)
    SizeBBottom,     (* size gadget uses bottom border *)

    Refresh0, Refresh1, (* Refresh bits *)

    BackDrop,        (* this is a BACKDROP window *)
    ReportMouseFlag, (* set this to hear about every mouse move *)
    GimmeZeroZero,   (* make extra border stuff *)
    Borderless,      (* set this to get a Window without border *)
    Activate,        (* when Window opens, it's the Active one *)

    (* Flags set by Intuition *)
    WindowActive,    (* this window is the active one *)
    InRequest,       (* this window is in request mode *)
    MenuState,       (* this Window is active with its Menus on *)

    (* Other User Flags *)
    RMBTrap,         (* Catch RMB events for your own *)
    NoCareRefresh,   (* not to be bothered with REFRESH *)

    Wi18, Wi19,      (* reserved *)
    Wi20, Wi21, Wi22, Wi23, (* reserved *)

    (* Other Intuition Flags *)
```

```

WindowRefresh,    (* Window is currently refreshing *)
WBenchWindow,     (* WorkBench tool ONLY Window *)

Wi26, Wi27, Wi28, Wi29, Wi30, Wi31  (* reserved *)
) ;

```

```
WindowFlagsSet = SET OF WindowFlags ;
```

```
TYPE
```

```
Window = RECORD
```

```

    NextWindow : WindowPtr ;      (* for the linked list in a screen *)
    LeftEdge, TopEdge : INTEGER ; (* screen dimensions of window *)
    Width, Height : INTEGER ;
    MouseY, MouseX : INTEGER ;    (* relative to upper-left of window *)
    MinWidth, MinHeight : INTEGER ; (* minimum sizes *)
    MaxWidth, MaxHeight : INTEGER ; (* maximum sizes *)
    Flags : WindowFlagsSet ;
    MenuStrip : MenuPtr ;         (* the strip of Menu headers *)
    Title : ADDRESS ;             (* the title text for this window *)
    FirstRequest : RequesterPtr ; (* all active Requesters *)
    DMRequest : RequesterPtr ;    (* double-click Requester *)
    ReqCount : INTEGER ;          (* count of reqs blocking Window *)
    WScreen : ScreenPtr ;         (* this Window's Screen *)
    RPort : RastPortPtr ;         (* this Window's own RasterPort *)
    BorderLeft, BorderTop,
    BorderRight, BorderBottom : BYTE ;
    BorderRPort : RastPortPtr ;
    FirstGadget : GadgetPtr ;
    Parent,
    Descendant : WindowPtr ;      (* for opening/closing the windows *)
    Pointer : ADDRESS ;           (* sprite data *)
    PtrHeight : BYTE ;            (* sprite height (not including
                                   sprite padding) *)
    PtrWidth : BYTE ;             (* sprite width (must be less than or
                                   equal to 16) *)
    XOffset, YOffset : BYTE ;     (* sprite offsets *)
    IDCMPFlags : IDCMPFlagsSet ;  (* User-selected flags *)
    UserPort, WindowPort : MsgPortPtr ;
    MessageKey : IntuiMessagePtr ;
    DetailPen, BlockPen : BYTE ;  (* for bar/border/gadget rendering *)
    CheckMark : ImagePtr ;

```

```

ScreenTitle : ADDRESS ;      (* if non-null, Screen title when
                               Window is active *)

GZZMouseX : INTEGER ;
GZZMouseY : INTEGER ;
GZZWidth : INTEGER ;
GZZHeight : INTEGER ;
ExtData : ADDRESS ;
UserData : ADDRESS ;        (* pointer to User data extension *)
END ; (* OF RECORD *)

```

(* Notes :

The border variables describe the window border. If you specify GimmeZeroZero when you open the window, then the upper-left of the ClipRect for this window will be upper-left of the BitMap (with correct offsets when in SuperBitMap mode; you MUST select GimmeZeroZero when using SuperBitMap). If you don't specify ZeroZero, then you save memory (no allocation of RastPort, Layer, ClipRect and associated Bitmaps), but you also must offset all your writes by BorderTop, BorderLeft and do your own mini-clipping to prevent writing over the system gadgets.

FirstGadget: You supply a linked-list of Gadgets for your Window. This list DOES NOT include system gadgets. You get the standard window system gadgets by setting flag-bits in the variable Flags.

Pointer: Sprite data information for your own Pointer set these AFTER you Open the Window by calling SetPointer().

CheckMark is a pointer to the imagery that will be used when rendering MenuItems of this Window that want to be checkmarked if this is equal to NULL, you'll get the default imagery.

GZZMouseX, GZZMouseY: These variables have the mouse coordinates relative to the inner-Window of GimmeZeroZero Windows. This is compared with the MouseX and MouseY variables, which contain the mouse coordinates relative to the upper-left corner of the Window, GimmeZeroZero notwithstanding.

GZZWidth, GZZHeight: These variables contain the width and height of the inner-Window of GimmeZeroZero Windows.

*)

NewWindow = RECORD

```

    LeftEdge, TopEdge : INTEGER ; (* screen dimensions of window *)
    Width, Height : INTEGER ;

```

```

DetailPen, BlockPen : BYTE ;(* for bar/border/gadget rendering *)
IDCMPFlags : IDCMPFlagsSet ;(* User-selected IDCMP flags *)
Flags : WindowFlagsSet ;    (* see Window record for details *)
FirstGadget : GadgetPtr ;
CheckMark : ImagePtr ;
Title : ADDRESS ;           (* the title text for this window *)
Screen : ScreenPtr ;
BitMap : BitMapPtr ;
MinWidth, MinHeight : INTEGER ;
MaxWidth, MaxHeight : INTEGER ;
Type : CARDINAL ;
END ; (* OF RECORD *)

```

(* Notes :

FirstGadget: You supply a linked-list of Gadgets for your Window. This list DOES NOT include system Gadgets. You get the standard system Window Gadgets by setting flag-bits in the variable Flags.

CheckMark is a pointer to the imagery that will be used when rendering MenuItems of this Window that want to be checkmarked if this is equal to NULL, you'll get the default imagery.

The Screen pointer is used only if you've defined a CUSTOMSCREEN and want this Window to open in it. If so, you pass the address of the Custom Screen structure in this variable. Otherwise, this variable is ignored and doesn't have to be initialized.

BitMap: If you have a SuperBitMap Window put the address of your BitMap structure in this variable. If not, this variable is ignored and doesn't have to be initialized.

Min and Max Width and Height: The values describe the minimum and maximum sizes of your Windows. These matter only if you've chosen the WindowSizing Gadget option, which means that you want to let the User to change the size of this Window. You describe the minimum and maximum sizes that the Window can grow by setting these variables. You can initialize any one these to zero, which will mean that you want to duplicate the setting for that dimension (if MinWidth == 0, MinWidth will be set to the opening Width of the Window). You can change these settings later using SetWindowLimits(). If you haven't asked for a SIZING Gadget, you don't have to initialize any of these variables.

The type variable describes the Screen in which you want this Window to open. The type value can either be CustomScreen or one of the system standard Screen Types such as WBenchScreen. See the type definitions under the Screen structure.

*)

(* Requesters *)

TYPE

RequesterFlags = (

 (* Flags set by the application *)

 PointRel, (* TopLeft is relative to pointer*)

 PreDrawn, (* ReqBMap points to predrawn Requester imagery *)

 Re2, Re3, Re4, Re5, Re6, Re7, Re8, Re9, Re10, (* rsrvd *)

 (* Flags set by Intuition *)

 ReqOffWindow, (* part of one of the Gadgets was offwindow *)

 ReqActive, (* this requester is active *)

 SysRequest, (* this requester caused by system *)

 DeferRefresh (* this Requester stops a Refresh broadcast *)

);

RequesterFlagsSet = SET OF RequesterFlags ;

Requester = RECORD

 OlderRequest : RequesterPtr ;

 LeftEdge, TopEdge : INTEGER ; (* dimensions of the entire box *)

 Width, Height : INTEGER ;

 RelLeft, RelTop : INTEGER ; (* for Pointer relativity offsets *)

 ReqGadget : GadgetPtr ; (* pointer to a list of Gadgets *)

 ReqBorder : BorderPtr ; (* the box's border *)

 ReqText : IntuitionTextPtr ; (* the box's text *)

 Flags : RequesterFlagsSet;

 BackFill : BYTE ; (*z pen number for back-plane fill
before draws *)

 ReqClipRect : ClipRect ;

 ImageBitMap : BitMapPtr ; (* points to the BitMap of
PreDrawn imagery *)

 ReqBitMap : BitMap ;

END ; (* OF RECORD *)

(* Notes :

The ClipRect and BitMap and used for rendering the requester.

If the BitMap plane pointers are non-zero, this tells the system that the image comes pre-drawn (if the appliprogram wants to define it's own box, in any shape or size it wants!); this is OK by Intuition as long as there's a good correspondence between the image and the specified Gadgets.

*)

(* Menus *)

TYPE

MenuFlags = (

 (* Flags set by both the application and intuition *)

 MenuEnabled, (* whether or not this menu is enabled *)

 Me1, Me2, Me3, Me4, Me5, Me6, Me7, (* Reserved *)

 (* Flags set by Intuition *)

 MenuItemsDrawn (* this menu's items are currently drawn *)

);

MenuFlagsSet = SET OF MenuFlags ;

(* Menu items *)

ItemFlags = (

 (* Flags set by the application *)

 CheckIt, (* whether to check this item if selected *)

 ItemText, (* set if textual, clear if graphical item *)

 CommSeq, (* set if there's an command sequence *)

 MenuToggle, (* set to toggle the check of a menu item *)

 ItemEnabled, (* set if this item is enabled *)

 It5, (* reserved *)

 (* these are the special highlight flag state meanings *)

 It6, It7,

 (* Flags set by both the application and Intuition *)

 Checked, (* if CheckIt, then set this when selected *)

 It9, (* reserved *)

```

It10,      (* reserved *)
It11,      (* reserved *)

(* Flags set by Intuition *)
IsDrawn,   (* this item's subs are currently drawn *)
HighItem,  (* this item is currently highlighted *)
MenuToggled (* this item was already toggled *)
) ;

```

```
ItemFlagsSet = SET OF ItemFlags ;
```

```
Menu = RECORD
```

```

    NextMenu : MenuPtr ;           (* same level *)
    LeftEdge, TopEdge : INTEGER ;  (* position of the select box *)
    Width, Height : INTEGER ;      (* dimensions of the select box *)
    Flags : MenuFlagsSet ;         (* see flag definitions below *)
    MenuName : ADDRESS ;           (* text for this Menu Header *)
    FirstItem : MenuItemPtr ;      (* pointer to first in chain *)

```

```
    (* these mysteriously-named variables are for internal use only *)
```

```
    JazzX, JazzY, BeatX, BeatY : INTEGER ;
```

```
END ; (* OF RECORD *)
```

```
MenuItem = RECORD
```

```

    NextItem : MenuItemPtr ;       (* pointer to next in list *)
    LeftEdge, TopEdge : INTEGER ;   (* position of the select box *)
    Width, Height : INTEGER ;       (* size of the select box *)
    Flags : ItemFlagsSet ;          (* see above *)
    MutualExclude : LONGINT ;       (* set bits mean this item
                                     excludes that item *)

    ItemFill : ADDRESS ;            (* points to Image, IntuitionText
                                     or NULL *)

```

```

(* when this item is pointed to by the cursor and the items highlight
mode HighImage is selected, this alternate image will be displayed *)

```

```

    SelectFill : ADDRESS ;          (* points to Image, IntuitionText
                                     or NULL *)

    Command : BYTE ;               (* only if application sets the
                                     CommSeq flag *)

    SubItem : MenuItemPtr ;        (* if non-zero, DrawMenu shows
                                     "-->" *)

```

(* The NextSelect field represents the menu number of next selected item (when user has drag-selected several items) *)

NextSelect : CARDINAL ;
END ; (* OF RECORD *)

(* Screens *)

TYPE

ScreenFlags = (
 (* Flags set by intuition *)
 ScreenType0, ScreenType1, ScreenType2, ScreenType3,

 ShowTitleFlag, (* this gets set by a call to ShowTitle() *)
 Beeping, (* set when Screen is beeping *)
 CustomBitMap, (* if you are supplying your own BitMap *)
 Sc7, Sc8 (* Rsrvd *)
);

ScreenFlagsSet = SET OF ScreenFlags ;

TYPE

Screen = RECORD
 NextScreen : ScreenPtr ; (* linked list of screens *)
 FirstWindow : ADDRESS (**WindowPtr**) ; (* list of Screen Windows *)
 LeftEdge, TopEdge : INTEGER ; (* parameters of the screen *)
 Width, Height : INTEGER ; (* parameters of the screen *)
 MouseY, MouseX : INTEGER ; (* position relative to upper-left *)
 Flags : ScreenFlagsSet ; (* see above *)
 Title : ADDRESS ; (* null-terminated Title text *)
 DefaultTitle : ADDRESS ; (* for Windows without ScreenTitle *)
 (* Bar sizes for this Screen and all Window's in this Screen *)
 BarHeight, BarVBorder, BarHBorder, MenuVBorder, MenuHBorder : BYTE ;
 WBorderTop, WBorderLeft, WBorderRight, WBorderBottom : BYTE ;
 Font : TextAttrPtr ; (* this screen's default font *)
 (* the display data structures for this Screen *)
 VPort : ViewPort ; (* describing the Screen's display *)
 RPort : RastPort ; (* describing Screen rendering *)
 BMap : BitMap ; (* auxiliary graphexcess baggage *)
 LInfo : LayerInfo ; (* each screen gets a LayerInfo *)
 FirstGadget : GadgetPtr ;
 DetailPen, BlockPen : BYTE ; (* for bar/border/gadget rendering *)

```

SaveColor0 : CARDINAL ;

BarLayer : LayerPtr ;      (* layer for Screen and Menu bars *)
ExtData : ADDRESS ;
UserData : ADDRESS ;      (* general-purpose User data *)
END ; (* OF RECORD *)

```

(* Notes :

ForstGadget: You supply a linked-list of Gadgets for your Screen. This list DOES NOT include system Gadgets. You get the standard system Screen Gadgets by default.

SaveColor0 : The variable is maintained by Intuition to support the DisplayBeep() color flashing technique.

*)

(* Gadgets *)

TYPE

```

GadgetFlags = (
    (* Flags set by the Application program *)
    Ga0,      (* Highlight options *)
    Ga1,
    GadgetImage,
    RelBottom, (* set if rel to bottom, clear if rel top *)
    RelRight,  (* set if rel to right, clear if to left *)
    RelWidth,
    RelHeight,
    Selected,
    Disabled
) ;

```

(* Notes :

GadgetImage set if the GadgetRender and SelectRender point to Image imagery, clear if it's a Border.

Combinations in RelBottom and RelRight specify to which corner the gadget's Left & Top coordinates are relative. If relative to Top/Left, these are "normal" coordinates (everything is relative to something in this universe)

RelWidth specifies that Width is relative to width of screen.

RelHeight specifies that Height is rel to height of screen.

Selected is initialized by the application and set by Intuition. It specifies whether or not this Gadget is currently selected/highlighted.

Disabled is initialized by the application and later set by Intuition according to your calls to On/OffGadget(). It specifies whether or not this Gadget is currently disabled from being selected.

*)

GadgetFlagsSet = SET OF GadgetFlags ;

(* The Activation flag bits *)

ActivationFlags = (

 RelVerify,
 GadgetImmediate,
 EndGadget,
 FollowMouse,
 RightBorder,
 LeftBorder,
 TopBorder,
 BottomBorder,
 ToggleSelect, (* this bit for toggle-select mode *)
 StringCenter,
 StringRight,
 LongInt, (* this String Gadget is actually LONG Int *)
 AltKeyMap (* this String has an alternate keypad *)
);

(* Notes :

RelVerify is set if you want to verify that the pointer was still over the gadget when the select button was released

GadgetImmediate when set, informs the caller that the gadget was activated when it was activated. this flag works in conjunction with the RelVerify flag

EndGadget, when set, tells the system that this gadget, when selected, causes the Requester or AbsMessage to be ended. Requesters or AbsMessages that are ended are erased and unlinked from the system

FollowMouse flag, when set, specifies that you want to receive reports on

mouse movements (ie, you want the ReportMouse function for your Window). When the Gadget is deselected (immediately if you have no RelVerify) the previous state of the ReportMouse flag is restored. You probably want to set the GadgetImmediate flag when using FollowMouse, since that's the only reasonable way you have of learning why Intuition is suddenly sending you a stream of mouse movement events. If you don't set RelVerify, you'll get at least one Mouse Position event.

If any of the border flags are set in a Gadget that's included in the Gadget list when a Window is opened, the corresponding Border will be adjusted to make room for the Gadget.

*)

ActivationFlagsSet = SET OF ActivationFlags ;

TYPE

Gadget = RECORD

```

    NextGadget : GadgetPtr ;           (* next gadget in the list *)
    LeftEdge, TopEdge : INTEGER ;      (* "hit box" of gadget *)
    Width, Height : INTEGER ;          (* "hit box" of gadget *)
    Flags : GadgetFlagsSet ;           (* see above *)
    Activation : ActivationFlagsSet;    (* see above *)
    GadgetType : CARDINAL ;             (* see above *)
    GadgetRender : ADDRESS ;            (* pointer to what to render,
                                         NULL if nothing *)
    SelectRender : ADDRESS;             (* pointer to highlighted struct *)
    GadgetText : IntuitionTextPtr ;    (* text for this gadget *)
    MutualExclude : LONGINT ;           (* set bits mean this gadget
                                         excludes that gadget,
                                         see below *)

    SpecialInfo : ADDRESS ;            (* pointer to special data for
                                         proportional, string, and
                                         integer gadgets *)

    GadgetID : CARDINAL ;              (* user-definable ID field *)
    UserData : ADDRESS ;               (* ptr to general purpose User data
                                         (ignored by Intuition) *)

END ; (* OF RECORD *)
```

(* Notes: By using the MutualExclude word, the appliprogram can describe which gadgets mutually-exclude which other ones. The bits in MutualExclude correspond to the gadgets in object containing the gadget list. If this gadget is selected and a bit is set in this gadget's MutualExclude and the

gadget corresponding to that bit is currently selected (e.g. bit 2 set and gadget 2 is currently selected) that gadget must be unselected. Intuition does the visual unselecting (with checkmarks) and leaves it up to the program to unselect internally *)

(* Miscellaneous *)

CONST

(* Menu stuff *)

NoMenu = 0001FH ;

NoItem = 0003FH ;

NoSub = 0001FH ;

MenuNULL = 0FFFFH ;

(* These definitions are for the CommSeq and CheckIT menu stuff. If CheckIT, I'll use a generic Width (for all resolutions) for the CheckMark. If COMMSEQ, likewise I'll use this generic stuff. *)

CheckWidth = 19 ;

CommWidth = 27 ;

LowCheckWidth = 13 ;

LowCommWidth = 16 ;

(* These are the AlertNumber definitions. If you are calling DisplayAlert() the AlertNumber you supply must have the ALERTTYPE bits set to one of these patterns. *)

AlertTYPE = 08000000H ;

RecoveryAlert = 00000000H ; (* the system can recover from this *)

DeadendAlert = 08000000H ; (* no recovery possible, this is it *)

(* When you're defining IntuiText for the Positive and Negative Gadgets created by a call to AutoRequest(), these definitions will get you reasonable looking text. The only field without a definition is the IText field; you decide what text goes with the Gadget. *)

AutoFrontPen = 0 ;

AutoBackPen = 1 ;

AutoDrawMode = Jam2 ;

AutoLeftEdge = 6 ;

AutoTopEdge = 3 ;

```
AutoITextFont = 0 (**NULL**);  
AutoNextText = 0 (**NULL**);
```

```
(* RawMouse Codes and Qualifiers (Console OR IDCMP) *)
```

```
CursorUp = 04CH ;  
CursorLeft = 04FH ;  
CursorRight = 04EH ;  
CursorDown = 04DH ;  
KeyCodeQ = 010H ;  
KeyCodeX = 032H ;  
KeyCodeN = 036H ;  
KeyCodeM = 037H ;
```

```
(* ----- *)
```

```
PROCEDURE AllocRemember ( VAR Key : RememberPtr ; Size : LONGCARD ;  
                          Flags : LONGWORD ) : ADDRESS ;  
(* AllocMem and create a link mode *)
```

```
PROCEDURE CurrentTime ( Seconds, Micros : ADDRESS ) ;  
(* Get the current time values *)
```

```
PROCEDURE DisplayAlert ( AlertNumber : CARDINAL ; VAR String: ARRAY OF CHAR ;  
                        Height : CARDINAL ) : BOOLEAN ;  
(* Create and display an alert *)
```

```
PROCEDURE DoubleClick ( StartSeconds, StartMicros, CurrentSeconds,  
                        CurrentMicros : LONGCARD ) : BOOLEAN ;  
(* Test two time values for double click timing *)
```

```
PROCEDURE DrawBorder ( VAR Ras : RastPort ; VAR Bor : Border ;  
                      LeftOffset, TopOffset : INTEGER ) ;  
(* Draws the specified border *)
```

```
PROCEDURE DrawImage ( VAR Ras : RastPort ; VAR Img : Image ;  
                     LeftOffset, TopOffset : INTEGER ) ;  
(* Draws the specified image *)
```

```
PROCEDURE FreeRemember ( VAR Key : RememberPtr ; ReallyForget : BOOLEAN ) ;  
(* Free memory allocated by AllocRemember *)
```

```
PROCEDURE IntuiTextLength ( VAR Text : IntuitionText ) : CARDINAL ;  
(* Get length of text *)
```



```

PROCEDURE PrintIText ( VAR Ras : RastPort ; VAR IText : IntuitionText ;
                      LeftEdge, RightEdge : INTEGER ) ;

(* Print the text *)

PROCEDURE RemakeDisplay () ;
(* Remake the entire Intuition display *)

PROCEDURE RethinkDisplay () ;
(* Global display reconstruction *)

(* ----- *)

(* Intuition internal procedures *)

PROCEDURE AlohaWorkbench ( VAR WBPort : MsgPort ) ;
(* Tell Intuition that workbench is saying hello/goodbye *)

PROCEDURE Intuition ( InputEvent : ADDRESS ) ;
(* Intuition main entry point. *)

END Intuition.

```

DEFINITION MODULE IO;

```
(** -----  
Commodore Amiga IO module  
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved  
----- **)
```

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 16-Dec-85

Version : 0.00a 16-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM Ports IMPORT Message;
FROM Devices IMPORT DevicePtr, UnitPtr;

TYPE

ioFlagSet = SET OF [0..7];

IORequestPtr = POINTER TO IORequest;

IORequest = RECORD

ioMessage: Message;

ioDevice: DevicePtr; (* device node pointer *)

ioUnit: UnitPtr; (* unit driver, private *)

ioCommand: CARDINAL; (* device command *)

ioFlags: ioFlagSet;

ioError: BYTE; (* error or warning number *)

END;

IOStdReq = RECORD

ioReq: IORequest; (* the standard IORequest structure *)

ioActual: LONGCARD; (* actual number of bytes transferred *)

ioLength: LONGCARD; (* requested number bytes transferred *)

ioData: ADDRESS; (* points to data area *)

ioOffset: LONGCARD; (* offset for block structured devices *)

END;

```

CONST
    StillInProgress = 0; (* NULL returned by CheckIO *)

CONST
    (* IO Errors *)
    IOOpenFail = -1; (* device/unit failed to open *)
    IOAborted = -2; (* request aborted *)
    IONoCmd = -3; (* command not supported *)
    IOBadLength = -4; (* not a valid length *)

CONST
    (* Device flags *)
    IOQuick = 0; (* bit 0 of ioFlags => do this io quickly *)

    (* Device commands *)
    CmdInvalid = 0;
    CmdReset = 1;
    CmdRead = 2;
    CmdWrite = 3;
    CmdUpdate = 4;
    CmdClear = 5;
    CmdStop = 6;
    CmdStart = 7;
    CmdFlush = 8;

    CmdNonStd = 9; (* non-standard device commands are from nine onwards *)

PROCEDURE AbortIO(VAR ioRequest: IORequest);
    (* halt the ioRequest if running.

       ioRequest: the I/O request to abort. *)

PROCEDURE BeginIO(VAR ioRequest: IORequest);
    (* Start I/O given a request block.

       ioRequest: the request to begin. *)

PROCEDURE CheckIO(VAR ioRequest: IORequest): IORequestPtr;
    (* get the I/O request status.

       ioRequest: 0 => still in progress, otherwise the address of the

```

device request block. *)

PROCEDURE DoIO(VAR ioRequest: IORequest): LONGINT;
(* perform an I/O command and wait for completion.

ioRequest: the request to start.

returns: 0 => completed OK, otherwise error number. *)

PROCEDURE SendIO(VAR ioRequest: IORequest);
(* initiate an I/O command - returns whether completed or not.

ioRequest: the request to start. *)

PROCEDURE WaitIO(VAR ioRequest: IORequest): LONGINT;
(* wait for completion of an I/O request.

ioRequest: the request to wait for; if completed, immediate return.

returns: 0 => completed OK, otherwise error number. *)

END IO.

DEFINITION MODULE KeyboardDevice;

(** -----

Commodore Amiga keyboard device module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 29-Jan-86

Version : 0.00a 29-Jan-86 Paul Curtis, TDI.
Original

*)

FROM IO IMPORT CmdNonStd;

CONST

KBDReadEvent = CmdNonStd + 0;

KBDReadMatrix = CmdNonStd + 1;

KBDAddResetHandler = CmdNonStd + 2;

KBDRemResetHandler = CmdNonStd + 3;

KBDResetHandlerDone = CmdNonStd + 4;

END KeyboardDevice.

DEFINITION MODULE Layers;

(** -----

Commodore Amiga layer module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 21-Jan-86

Version : 0.00a 21-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM GraphicsLibrary IMPORT BitMapPtr, Rectangle;
FROM Rasters IMPORT RastPort, RastPortPtr;
FROM Ports IMPORT MsgPort, Message;
FROM Regions IMPORT RegionPtr;
FROM Tasks IMPORT TaskPtr;

TYPE

LayerPtr = POINTER TO Layer;
ClipRectPtr = POINTER TO ClipRect;
LayerInfoPtr = POINTER TO LayerInfo;

TYPE

Layer = RECORD
 front: LayerPtr;
 back: LayerPtr;
 ClipRect: ClipRectPtr;
 rp: RastPortPtr;
 bounds: Rectangle;
 Lock: BYTE;
 LockCount: BYTE;
 LayerLockCount: BYTE;
 reserved: BYTE;

```

reserved1: CARDINAL;
Flags: BITSET;
SuperBitMap: BitMapPtr;
SuperClipRect: ClipRectPtr;
Window: ADDRESS;
ScrollX: INTEGER;
ScrollY: INTEGER;
LockPort: MsgPort;
LockMessage: Message;
ReplyPort: MsgPort;
lLockMessage: Message;
DamageList: RegionPtr;
cliprects: ClipRectPtr;
p1: ADDRESS;
END;

```

TYPE

```

ClipRect = RECORD
    next: ClipRectPtr;
    prev: ClipRectPtr;
    lobs: LayerPtr;
    bitMap: BitMapPtr;
    bounds: Rectangle;
    p1: ClipRectPtr;
    p2: ClipRectPtr;
    reserved: LONGINT;
END;

```

TYPE

```

LayerInfo = RECORD
    topLayer: LayerPtr;
    checkLP: LayerPtr; (* system only *)
    obs: LayerPtr; (* system only *)
    RPRReplyPort: MsgPort; (* for rastport locking *)
    LockPort: MsgPort; (* for screen locking *)
    Lock: BYTE;
    broadcast: BYTE; (* bunch of messages sent *)
    LockNest: BYTE;
    pad: BYTE;
    Locker: TaskPtr;
    bytereserved: ARRAY [0..1] OF BYTE;

```

```

wordreserved: ARRAY [0..1] OF CARDINAL;
longreserved: ARRAY [0..1] OF LONGCARD;
END;

```

```

PROCEDURE ClipBlit(VAR src,dst: RastPort; sx,sy, dx,dy, xs,ys, m: CARDINAL);
(* calls BltBitMap after accounting for windows.

```

```

    src: raster port for the source of the blit.
    dst: raster port to receive the blitted data.
    sx,sy: the top left offset into src for the data.
    dx,dy: the top left offset into dst for the data.
    xs: width of the blit.
    ys: height of the blit.
    m: the blitter mode. *)

```

```

PROCEDURE CopySBitMap(VAR layer: Layer; li: ADDRESS);
(* synchronise layer window with contents of super bitmap.

```

```

    layer: layer to synchronise. *)

```

```

PROCEDURE LockLayerRom(VAR layer: Layer);
(* lock layer structure by ROM code.

```

```

    layer: the layer to lock. *)

```

```

PROCEDURE SyncSBitMap(VAR layer: Layer);
(* synchronise super bitmap with whatever is in standard layer bounds.

```

```

    layer: the layer to synchronise. *)

```

```

PROCEDURE UnlockLayerRom(VAR layer: Layer);
(* unlock layer by ROM code.

```

```

    layer: the layer to unlock. *)

```

```

END Layers.

```


DEFINITION MODULE Libraries;

(** -----

Commodore Amiga libraries module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 16-Dec-85

Version : 0.00a 16-Dec-85 Paul Curtis, TDI,
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;

FROM Nodes IMPORT Node;

CONST

LibVecSize = 6; (* six bytes for a library vector entry *)

LibReserved = 4; (* four vectors reserved: Open, close, Expunge, ExtFunc *)

LibBase = -LibVecSize;

LibUserDef = LibBase-LibReserved*LibVecSize; (* -30 *)

LibNonstd = LibUserDef;

CONST

(* Vector offsets from library base *)

LibOpen = -6; (* standard library open offset *)

LibClose = -12; (* standard library close offset *)

LibExpunge = -18; (* standard library expunge offset *)

LibExtFunc = -24;

TYPE

LibStates = (LIBFSumming, (* currently checksumming *)

LIBFChanged, (* just changed the library *)

LIBFSumUsed, (* set if we should bother to sum *)

LIBFDelexp); (* delayed expunge *)

LibStateSet = SET OF LibStates;

```

LibraryPtr = POINTER TO Library;
Library = RECORD
    libMode: Mode;
    libFlags: LibStateSet;
    libpad: BYTE;
    libNegSize: CARDINAL; (* nr. bytes before library *)
    libPosSize: CARDINAL; (* number of bytes after library *)
    libVersion: CARDINAL;
    libRevision: CARDINAL;
    libIdString: POINTER TO ARRAY [0..32767] OF CHAR;
    libSum: CARDINAL; (* library checksum *)
    libOpenCnt: CARDINAL; (* number of current opens *)
END;

```

```

PROCEDURE AddLibrary(library: LibraryPtr);
(* add a library to the system.

```

library: the library to add, a properly initialised Library record. *)

```

PROCEDURE CloseLibrary(library: LibraryPtr);
(* conclude access to a library.

```

library: the library pointer, obtained by OpenLibrary, that should be closed. *)

```

PROCEDURE MakeLibrary(vectors, structure, init: ADDRESS; dataSize: LONGCARD;
    segList: ADDRESS): LibraryPtr;
(* construct a library.

```

vectors: pointer to array of function pointers or function displacements.
If the first word of the array is -1, then the array contains relative word displacements, otherwise the array contains absolute function pointers.

structure: points to InitStruct data region. If 0, then it will not be called.

init: a procedure to call before adding the library to the system. If null, it will not be called.

dataSize: the size of the library data area, including the standard library node data.

segList: pointer to the memory segment list, used by DOS.

returns: reference address of library. *)

PROCEDURE OpenLibrary(VAR libName: ARRAY OF CHAR; vers: LONGCARD): LibraryPtr;
(* gain access to a library.

libName: the name of the library to open, null terminated.

vers: the version of the library required.

returns: 0 => library not available, otherwise a library pointer for
the library. *)

PROCEDURE RemLibrary(library: LibraryPtr): LONGCARD;
(* remove a library from the system.

library: pointer to library node structure that should be removed.

returns: 0 => removed ok, otherwise an error number. *)

PROCEDURE SetFunction(library: LibraryPtr; funcOffset: CARDINAL;
func: PROC): ADDRESS;
(* change a function vector in a library.

library: the library to be changed.

funcOffset: the offset into the library for the function to be changed.

func: the new procedure to replace the old one.

returns: the old library function implementation address. *)

PROCEDURE SumLibrary(library: LibraryPtr);
(* compute and check the checksum of a library.

library: the library to be checksummed. If the checksum computed
differs from the old checksum an alert occurs. *)

END Libraries.

DEFINITION MODULE Lists;

(** -----

Commodore Amiga list module

(c) Copyright 1985, 1985 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 11-Dec-85

Version : 0.00a 11-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;

FROM Nodes IMPORT NodePtr;

TYPE

ListPtr = POINTER TO List;

List = RECORD

lhHead: NodePtr; (* head of the list *)

lhTail: NodePtr; (* tail of the list *)

lhTailPred: NodePtr; (* predecessor or tail *)

lhType: BYTE; (* type of list *)

lPad: BYTE;

END;

PROCEDURE AddHead(list: ListPtr; node: NodePtr);

(* insert a node at the head of a list.

list: the target list header.

node: the node to insert at head. *)

PROCEDURE AddTail(list: ListPtr; node: NodePtr);

(* append a node to the tail of a list.

list: the target list header.

node: the node to append to the tail. *)

PROCEDURE Enqueue(list: ListPtr; node: NodePtr);
(* insert or append a node to a system queue.

list: the system queue header.

node: the node to enqueue - insert is based on node priority.

New nodes are inserted in front of nodes with lower priority,
nodes of equal priority are a FIFO queue. *)

PROCEDURE FindName(start: ADDRESS; VAR name: ARRAY OF CHAR): NodePtr;
(* find a system list node with a given name.

start: a list header or list node to start the search. If a node,
this one is skipped.

name: the name of the node required, null terminated.

returns: 0 => node not found, otherwise the node with the same name. *)

PROCEDURE Insert(list: ListPtr; node, listNode: NodePtr);
(* insert a node into a list.

list: the target list header.

node: the node to insert after listNode.

listNode: the node after which to insert node. *)

PROCEDURE NewList(list: ListPtr);
(* initialise a list header.

list: the list header to initialise to the empty list. *)

PROCEDURE RemHead(list: ListPtr): NodePtr;
(* remove the head node from a list.

list: the target list header from which to remove the head node.

returns: 0 => list is empty, otherwise pointer to the node removed
from the head of the list. *)

PROCEDURE Remove(list: ListPtr; node: NodePtr);
(* remove a node from a list.

list: the target list header.
node: the node to remove. *)

PROCEDURE RemTail(list: ListPtr): NodePtr;

(* remove the tail node from a list.

list: the node header from which to remove the tail node.

returns: 0 => list is empty, otherwise pointer to the node removed
from the tail of the list. *)

END Lists.

DEFINITION MODULE MathLib0;

(* Provides standard maths functions.

These math functions are accurate to the seventh digit. The eighth digit is dubious as a 23 bit fractional mantissa is used in the system. Where applicable, procedure names and parameters are compatible with those suggested by N Wirth in the book 'Programming in MODULA-2'.

Copyright (c) 1984 TDI Ltd and
Robinson Systems Engineering Ltd

Original Authors : P Curtis, R Hartell, RSE Ltd

Version : 1.00a 10-Aug-84 RSE

Modifications :

*)

EXPORT QUALIFIED

(* CONST *)	pi, e,	
(* PROCS *)	RadToDeg, DegToRad, real, entier,	(* Conversions *)
	sin, cos, tan, arctan,	(* Transcendentals *)
	exp, ln, log,	(* Logarithms *)
	power, sqrt;	(* Power functions *)

CONST

pi = 3.1415926536;
e = 2.7182818284;

(* Conversions *)

PROCEDURE RadToDeg (RadianAngle : REAL) : REAL;
(* Convert the given angle in radians to degrees *)

PROCEDURE DegToRad (DegreeAngle : REAL) : REAL;
(* Convert the given angle in degrees to radians *)

PROCEDURE real (x : INTEGER) : REAL;
(* Convert an integer value to a real value *)

PROCEDURE entier (x : REAL) : INTEGER;

(* Convert a real value to an integer value by truncation *)

(* Transcendentals *)

PROCEDURE sin (x : REAL) : REAL;

(* Evaluate the sine of x radians *)

PROCEDURE cos (x : REAL) : REAL;

(* Evaluate the cosine of x radians *)

PROCEDURE tan (x : REAL) : REAL;

(* Evaluate the tangent of x radians *)

PROCEDURE arctan (x : REAL) : REAL;

(* Evaluate the inverse tangent of x radians *)

(* Logarithms *)

PROCEDURE exp (x : REAL) : REAL;

(* Evaluate the exponential of x *)

PROCEDURE ln (x : REAL) : REAL;

(* Evaluate the natural logarithm of x *)

PROCEDURE log (x : REAL) : REAL;

(* Evaluate the common logarithm of x (to the base 10) *)

(* Power functions *)

PROCEDURE power (x, y : REAL) : REAL;

(* Raise x to the y th power *)

PROCEDURE sqrt (x : REAL) : REAL;

(* Evaluate the square root of x using range reduction *)

END (* DEFINITION MODULE *) MathLib0.

DEFINITION MODULE Memory;

(** -----

Commodore Amiga memory module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 04-Dec-85

Version : 0.00a 04-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;

FROM Nodes IMPORT Node;

TYPE

MemReqSet = SET OF [0..31];

CONST

(* Memory requirement types *)

MemPublic = 0; (* non relocatable ram *)

MemChip = 1; (* ram available to device chips *)

MemFast = 2; (* fast offboard memory *)

MemClear = 16; (* clear memory before returning *)

MemLargest = 17; (* largest available memory chunk *)

MemFailed = 31; (* not enough memory, returned by AllocEntry *)

TYPE

MemChunkPtr = POINTER TO MemChunk;

MemChunk = RECORD

mcNext: MemChunkPtr;

mcBytes: LONGCARD;

END;

MemHeaderPtr = POINTER TO MemHeader;

MemHeader = RECORD

```

mhNode: Node;
mhAttributes: CARDINAL; (* characteristics of this region *)
mhFirst: MemChunkPtr; (* first free region *)
mhLower: ADDRESS; (* lower memory bound *)
mhUpper: ADDRESS; (* upper memory bound+1 *)
mhFree: LONGCARD; (* total number of free bytes *)
END;

```

MemEntryPtr = POINTER TO MemEntry;

MemEntry = RECORD

```

    MeUn: RECORD CASE BOOLEAN OF
        TRUE:
            meuRegs: MemReqSet; | (* the AllocMem requirements *)
        FALSE:
            meuAddr: ADDRESS; (* addr of this mem region *)
    END;
    END;
    meLength: LONGCARD;
END;

```

MemListHeaderPtr = POINTER TO MemListHeader;

MemListHeader = RECORD

```

    mlNode: Node;
    mlNumEntries: CARDINAL; (* nr. entries following *)
END;

```

PROCEDURE Allocate(freeList: ADDRESS; byteSize: LONGCARD): ADDRESS;
 (* allocate a block of memory.

freeList: the free list header in which this block will be allocated.
 byteSize: the size of the desired block in bytes.

returns: 0 => no free regions, memory not allocated, otherwise
 a pointer to the memory block. This will be longword
 aligned to 8 bytes. *)

PROCEDURE AllocEntry(memList: ADDRESS): ADDRESS;
 (* allocate many regions of memory.

memList: pointer to a MemList structure.

returns: a new MemList structure filled with the memory that

has been allocated. If bit 31 is set, i.e. the MemFailed bit, then an area of memory was not allocated. The type of memory that was not allocated may be found by stripping off the MemFailed bit. *)

PROCEDURE AllocMem(byteSize: LONGCARD; reqs: MemReqSet): ADDRESS;
(* allocate memory given certain requirements.

byteSize: the size of the required block in bytes. It is rounded up to a multiple of eight bytes.

reqs: the requirements for the allocated memory block.

returns: 0 => no free memory available, otherwise the base address of the memory block allocated. *)

PROCEDURE AvailMem(reqs: MemReqSet): LONGCARD;
(* memory available given certain requirements.

reqs: the types of memory to be considered.

returns: the total amount of memory free for those requirements. *)

PROCEDURE Deallocate(freeList, memBlk: ADDRESS; byteSize: LONGCARD);
(* deallocate a block of memory.

freeList: pointer to the free list.

memBlk: memory block to return.

byteSize: the size of the memBlk in bytes. *)

PROCEDURE FreeEntry(memList: ADDRESS);
(* free many regions of memory.

memList: pointer to MemList structure, filled by AllocEntry. *)

PROCEDURE FreeMem(memBlk: ADDRESS; byteSize: LONGCARD);
(* free memory with knowledge.

memBlk: the address of the memory block to free, returned by AllocMem.

byteSize: the size of the memory block, in bytes. *)

END Memory.

DEFINITION MODULE Menus ;

(* -----

TDI Modula-2/Amiga : Intuition/Menus

----- *)

(* ----- *)

(* (c) Copyright TDI Software Ltd. 1986. All Rights Reserved *)

(* ----- *)

FROM SYSTEM IMPORT ADDRESS, BYTE ;

FROM Intuition IMPORT ItemFlagsSet, ItemFlags, Menu, Window ;

(* NB. The Intuition library must be loaded before calling this module
(see Intuition.def), *)

CONST

(* Compound flags *)

HighFlags = ItemFlagsSet{It6,It7} ;

HighImage = ItemFlagsSet{} ; (* use the user's "select image" *)

HighComp = ItemFlagsSet{It6} ; (* complement the selectbox *)

HighBox = ItemFlagsSet{It7} ; (* box the selectbox *)

HighNone = ItemFlagsSet{It6,It7} ; (* don't highlight *)

PROCEDURE ClearMenuStrip (VAR Win : Window) ;

(* Clears the menu strip from a window *)

PROCEDURE ItemAddress (VAR MenuStrip : Menu ; MenuNumber : INTEGER) :ADDRESS;

(* Returns the address of the specified menu item *)

PROCEDURE OffMenu (VAR Win : Window ; MenuNumber : INTEGER) ;

(* Disables the menu or menu item *)

PROCEDURE SetMenuStrip (VAR Win : Window ; VAR Men : Menu) ;

(* Attaches the menu strip to a window *)

END Menus.

DEFINITION MODULE NarratorDevice;

(**

Commodore Amiga narrator module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

**)-----

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 22-Dec-85

Version : 0.00a 22-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;

FROM IO IMPORT IOStdReq;

CONST

NarratorName = "narrator.device";

CONST

(* Error Codes *)

NDNoMem = -2; (* can't allocate memory *)

NDNoAudLib = -3; (* can't open audio device *)

NDMakeBad = -4; (* error in MakeLibrary call *)

NDUnitErr = -5; (* unit is not 0 *)

NDCanAlloc = -6; (* can't allocate audio channel(s) *)

NDUnimpl = -7; (* unimplemented command *)

NDNoWrite = -8; (* read for mouth without write first *)

NDExpunged = -9; (* can't open, deferred expunge bit set *)

NDPhonErr = -20; (* phoneme code spelling error *)

NDRateErr = -21; (* rate out of bounds *)

NDPitchErr = -22; (* pitch out of bounds *)

NDSexErr = -23; (* sex not valid *)

NDModeErr = -24; (* mode not valid *)

NDFreqErr = -25; (* sampling frequency out of bounds *)

NDVolErr = -26; (* volume out of bounds *)

TYPE

Sex = CARDINAL; (* sex of voice *)

CONST

Male = 0;

Female = 1;

TYPE

PitchMode = CARDINAL; (* how to inflex *)

CONST

Natural = 0;

Robotic = 1;

CONST

MinPitch = 65; (* minimum pitch, 65 Hz *)

MaxPitch = 320; (* maximum pitch, 320 Hz *)

MinRate = 40; (* minimum speaking rate, 40 wpm *)

MaxRate = 400; (* maximum speaking rate, 400 wpm *)

MinFreq = 5000; (* minimum sampling frequency, 5000 Hz *)

MaxFreq = 28000; (* maximum sampling frequency, 28000 Hz *)

MinVol = 0; (* minimum volume *)

MaxVol = 64; (* maximum volume *)

TYPE

VolRange = [MinVol..MaxVol];

FreqRange = [MinFreq..MaxFreq];

RateRange = [MinRate..MaxRate];

PitchRange = [MinPitch..MaxPitch];

CONST

(* Defaults *)

DefPitch = 110; (* Default pitch *)

DefRate = 150; (* Default speaking rate, words per minute *)

DefVol = 64; (* Default volume, full *)

DefFreq = 22200; (* Default sampling frequency, Hertz *)

DefSex = Male; (* Default sex *)

DefMode = Natural; (* Default mode *)

```
NoVolume = MinVol; (* no speech *)
```

```
TYPE
```

```
  NarratorRB = RECORD
```

```
    message: IOStdReq; (* standard IORB *)
    rate: RateRange; (* speaking rate (words/minute) *)
    pitch: PitchRange; (* baseline pitch in Hertz *)
    mode: PitchMode; (* pitch mode *)
    sex: Sex; (* sex of voice *)
    chMasks: ADDRESS; (* ptr to audio allocation maps *)
    nmMasks: CARDINAL; (* nr. audio allocation maps *)
    volume: VolRange; (* volume *)
    sampFreq: FreqRange; (* audio sampling freq *)
    mouths: BYTE; (* <> 0 => generate mouths *)
    chanMask: BYTE; (* which channel mask used *)
    numChan: BYTE; (* nr. channel masks used *)
    pad: BYTE;
  END;
```

```
  MouthRB = RECORD
```

```
    voice: NarratorRB; (* speech IORB *)
    width: BYTE; (* width, returned value *)
    height: BYTE; (* height, returned value *)
    shape: BYTE; (* internal use *)
    pad: BYTE;
  END;
```

```
END NarratorDevice.
```

DEFINITION MODULE Nodes;

(** -----

Commodore Amiga node module

(c) Copyright 1985, 1985 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 11-Dec-85

Version : 0.00a 11-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS, BYTE;

TYPE

NodeType = (NTUnknown, NTTask, NTInterrupt, NTDevice, HTMsgPort,
NTMessage, NTFreeMsg, NTReplyMsg, NTResource, NTLibrary,
NTMemory, NTSoftInt, NTFont, NTProcess);

NodePtr = POINTER TO Node;

Node = RECORD

InSucc: NodePtr; (* the next node on the list *)

InPred: NodePtr; (* the previous node on the list *)

InType: BYTE; (*NodeType*) (* type of node this is *)

InPri: BYTE; (*[-128..127]*) (* the node priority *)

InName: ADDRESS; (* the nodes name, 0 => no name *)

END;

END Nodes.

DEFINITION MODULE ParallelDevice;

(*-----

Commodore Amiga parallel device module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

-----*)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 30-Dec-85

Version : 0.00a 30-Dec-85 Paul Curtis, TDI.
Original

*)

FROM IO IMPORT IOStdReq, CmdNonStd;

CONST

ParallelName = "parallel.device";

CONST

PDCmdQuery = CmdNonStd + 0;

PDCmdSetParams = CmdNonStd + 1;

CONST

(* parallel device errors *)

ParErrDevBusy = 1;

ParErrBufTooBig = 2;

ParErrInvParam = 3;

ParErrLineErr = 4;

ParErrNotOpen = 5;

ParErrPortReset = 6;

ParErrInitErr = 7;

TYPE

IOPArray = RECORD

PTermArray0: ARRAY [0..3] OF CHAR;

PTermArray1: ARRAY [0..3] OF CHAR;

END;

TYPE

```
ParFlagSet = SET OF [0..7];  
ParStatus = (PrinterSelected, PaperOut, PrinterBusy, Writing);  
ParStatusSet = SET OF ParStatus;
```

TYPE

```
IOExtPar = RECORD  
    ioPar: IOStdReq;  
    ioPWBuflen: LONGCARD; (* parallel port write buffer, bytes *)  
    ioStatus: ParStatusSet; (* parallel port and register status *)  
    ioParFlags: ParFlagSet;  
    ioPTermArray: IOArray; (* termination character array *)  
END;
```

CONST

```
ParShared = 5; (* non-exclusive access if set *)  
ParEOFMode = 1; (* EOF mode enabled *)
```

CONST

```
(* ioFlags *)  
IOParQueued = 6; (* request queued *)  
IOParAbort = 5; (* request aborted *)  
IOParActive = 4; (* request is queued, or is active *)
```

END ParallelDevice.

DEFINITION MODULE Pens;

(** -----

Commodore Amiga graphic pens module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 16-Jan-86

Version : 0.00a 16-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT WORD;

FROM GraphicsLibrary IMPORT DrawingModeSet;

FROM Rasters IMPORT RastPort;

PROCEDURE Draw(VAR rp: RastPort; x,y: CARDINAL);

(* draw a line from the current pen position and the new x,y position.

rp: the rastport to draw into.

x,y: the new pen position; also end point of line. *)

PROCEDURE Flood(VAR rp: RastPort; mode: CARDINAL; x,y: CARDINAL);

(* flood fill enclosed area.

rp: the rastport to fill.

mode: 0 => boundary fill, 1 => flood fill.

x,y: the point to start the fill from. *)

PROCEDURE Move(VAR rp: RastPort; x,y: CARDINAL);

(* move graphic pen position.

rp: the rastport to move the pen in.

x,y: the new pen position. *)

```

PROCEDURE PolyDraw(VAR rp: RastPort; count: LONGCARD; VAR pts: ARRAY OF WORD);
  (* draw lines from table of x,y values.

      rp: the rastport to draw into.
      count: the number of lines in the pts array.
      pts: the array of x,y coordinate pairs. Each pair is two CARDINAL
           values. *)

PROCEDURE ReadPixel(VAR rp: RastPort; x,y: CARDINAL): CARDINAL;
  (* read pen number from pixel x,y in rp.

      rp: the rastport to read from.
      x,y: the point to read from.

      returns: the pen number at x,y. *)

PROCEDURE RectFill(VAR rp: RastPort; xmin,ymin, xmax,ymax: CARDINAL);
  (* fill a rectangle.

      xmin,ymin: the upper left corner of the rectangle.
      xmax,ymax: the lower right corner of the rectangle. *)

PROCEDURE SetAPen(VAR rp: RastPort; pen: CARDINAL);
  (* set primary pen.

      rp: the rastport to set the primary pen of.
      pen: the new pen value. *)

PROCEDURE SetBPen(VAR rp: RastPort; pen: CARDINAL);
  (* set secondary pen.

      rp: the rastport to set the secondary pen of.
      pen: the new pen value. *)

PROCEDURE SetDrMd(VAR rp: RastPort; mode: DrawingModeSet);
  (* set drawing mode.

      rp: the rastport to set the drawing mode of.
      mode: the new drawing mode. *)

PROCEDURE WritePixel(VAR rp: RastPort; x,y: CARDINAL);

```

(* change the pen number of one pixel.

rp: the rastport in which to change a pixel.
x,y: the point to set. *)

END Pens.

DEFINITION MODULE PenUtils;

(** -----

Commodore Amiga pen utilities module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 27-Jan-86

Version : 0.00a 27-Jan-86 Paul Curtis, TDI.
Original

*)

FROM Rasters IMPORT RastPort;

FROM SYSTEM IMPORT WORD;

PROCEDURE SetOPen(VAR rp: RastPort; pen: CARDINAL);

(* set outline pen colour and outline mode.

rp: the rastport to set the 0 pen colour of.

pen: the colour to set the 0 pen to. *)

PROCEDURE SetDrPt(VAR rp: RastPort; VAR pat: BITSET);

(* set the line drawing pattern.

rp: the rastport to set the line drawing pattern of.

pat: the pattern to use. *)

PROCEDURE SetWrMsk(VAR rp: RastPort; VAR mask: BITSET);

(* set the plane write protect mask.

rp: the rastport to set the mask for.

mask: the mask to set. *)

PROCEDURE SetAfPat(VAR rp: RastPort; VAR pat: ARRAY OF WORD; n: CARDINAL);

(* set the fill pattern.

rp: the rastport to set the fill pattern for.
pat: the pattern to set.
n: 2^n = number of lines in pattern. *)

PROCEDURE BoundaryOff(VAR rp: RastPort);

(* turns boundary mode off when filling.

rp: the rastport to turn boundary mode off in. *)

END PenUtils.

DEFINITION MODULE Ports;

(** -----

Commodore Amiga ports module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 16-Dec-85

Version : 0.00a 16-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE;
FROM Tasks IMPORT TaskPtr;
FROM Nodes IMPORT Node;
FROM Lists IMPORT List;

TYPE

MsgType = (PASignal, PASoftInt, PAIgnore, PFAction);

MsgPortPtr = POINTER TO MsgPort;

MsgPort = RECORD

mpNode: Node;

mpFlags: BYTE; (* MsgType *)

mpSigBit: BYTE; (* signal bit nr. *)

mpSigTask: TaskPtr; (* task to be signalled *)

mpMsgList: List; (* message linked list *)

END;

MessagePtr = POINTER TO Message;

Message = RECORD

mnNode: Node;

mnReplyPort: MsgPortPtr; (* message reply port *)

mnLength: CARDINAL; (* message length in bytes *)

END;


```

PROCEDURE AddPort(port: MsgPortPtr);
  (* add a message port to the system.

     port: the message port to add to the system. *)

PROCEDURE FindPort(VAR name: ARRAY OF CHAR): MsgPortPtr;
  (* find a given system message port.

     name: the name of the port to find.

     returns: 0 => no port found, otherwise pointer to message
              port structure. *)

PROCEDURE GetMsg(port: MsgPortPtr): MessagePtr;
  (* get next message from a message port.

     port: the receiver message port.

     returns: 0 => no messages, otherwise pointer to the first
              message available. *)

PROCEDURE PutMsg(port: MsgPortPtr; message: MessagePtr);
  (* put a message to a message port.

     port: the port to send the message to.
     message: the message to send to the port. *)

PROCEDURE RemPort(port: MsgPortPtr);
  (* remove a message port from the system.

     port: the message port to remove. *)

PROCEDURE ReplyMsg(message: MessagePtr);
  (* put a message to its reply port.

     message: the message to send. *)

PROCEDURE WaitPort(port: MsgPortPtr): MessagePtr;
  (* wait for a given port to be non-empty.

     port: the port to wait for a message from.

```

returns: pointer to a returned message. X)

END Ports.

DEFINITION MODULE PortUtils;

(** -----

Commodore Amiga port utilities module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 16-Dec-85

Version : 0.80a 16-Dec-85 Paul Curtis, TDI,
Original

*)

FROM Ports IMPORT MsgPortPtr;

PROCEDURE CreatePort(VAR name: ARRAY OF CHAR; pri: INTEGER): MsgPortPtr;
(* create a message port.

name: the name of the port. If it is a null string (i.e. ""), the
port is local and closely coupled tasks may pass messages
through it. If it is not null, the port is open to all
system tasks to pass messages, as FindPort will be able
to find this port.

pri: the node priority, ranging from -128 to 127. Zero is the
normal priority.

returns: 0 => port could not be made, otherwise a pointer to
an initialised message port. *)

PROCEDURE DeletePort(port: MsgPortPtr);
(* delete a message port.

port: the message port to delete, created by CreatePort. *)

END PortUtils.

DEFINITION MODULE Preferences ;

(* -----

TDI Modula-2/Amiga : Intuition/Preferences

----- *)

(* ----- *)

(* (c) Copyright TDI Software Ltd. 1986. All Rights Reserved *)

(* ----- *)

FROM SYSTEM IMPORT BYTE, ADDRESS ;

FROM TimerDevice IMPORT TimeVal;

(* NB. The Intuition library must be loaded before calling this module
(see Intuition.def). *)

CONST

(* these are the definitions for the printer configurations *)

FileNameSize = 30 ; (* Filename size *)

PointerSize = (1 + 16 + 1) * 2 ; (* Size of Pointer data buffer *)

(* These constants are for the default font size. These actually describe the height of the default fonts. The default font type is the topaz font, which is a fixed width font that can be used in either eighty-column or sixty-column mode. The Preferences structure reflects which is currently selected by the value found in the variable FontSize, which may have either of the values defined below. These values actually are used to select the height of the default font. By changing the height, the resolution of the font changes as well.

*)

TopazEighty = 8 ;

TopazSixty = 9 ;

TYPE

Preferences = RECORD

FontHeight : BYTE ; (* height for system default font *)

PrinterPort : BYTE ; (* printer port connection *)

BaudRate : BYTE ; (* baud rate for the serial port *)

KeyRptSpeed : TimeVal ; (* repeat speed for keyboard *)

```

KeyRptDelay : TimeVal ; (* Delay before keys repeat      *)
DoubleClick : TimeVal ; (* Interval allowed between clicks *)
PointerMatrix : ARRAY[0..PointerSize-1] OF CARDINAL ;
                                (* Definition of pointer sprite *)
XOffset : BYTE ;              (* X-Offset for active 'bit' *)
YOffset : BYTE ;              (* Y-Offset for active 'bit' *)
color17 : CARDINAL ;          (* Colours for sprite pointer *)
color18 : CARDINAL ;
color19 : CARDINAL ;
PointerTicks : CARDINAL;(* Sensitivity of the pointer      *)
color8 : CARDINAL ;          (* workbench screen colors *)
color1 : CARDINAL ;
color2 : CARDINAL ;
color3 : CARDINAL ;
ViewXOffset : BYTE ;         (* Offset for top lefthand corner *)
ViewYOffset : BYTE ;         (* X and Y dimensions *)
ViewInitX,
ViewInitY : CARDINAL ; (* View initial offset values *)
EnableCLI : CARDINAL ; (* CLI availability switch *)
PrinterType : CARDINAL ;(* printer type *)
PrinterFilename : ARRAY [0..FileNameSize-1] OF CHAR ;
                                (* file for printer *)
PrintPitch : CARDINAL ; (* print pitch *)
PrintQuality : CARDINAL;(* print quality *)
PrintSpacing : CARDINAL;(* number of lines per inch *)
PrintLeftMargin : CARDINAL ; (* left margin in characters *)
PrintRightMargin : CARDINAL; (* right margin in characters *)
PrintImage : CARDINAL ; (* positive or negative *)
PrintAspect : CARDINAL ;(* horizontal or vertical *)
PrintShade : CARDINAL ; (* b&w, half-tone, or color *)
PrintThreshold : CARDINAL ; (* darkness ctrl for b/w dumps *)
PaperSize : CARDINAL ; (* paper size *)
PaperLength : CARDINAL ;(* paper length in number of lines *)
PaperType : CARDINAL ; (* continuous or single sheet *)
padding : ARRAY [0..49] OF CHAR ;(* For system expansion *)
END ; (* OF RECORD *)

```

CONST

```

(* PrinterPort *)
ParallelPrinter = 000H ;

```

SerialPrinter = 001H ;

(* BaudRate *)

Baud110 = 000H ;

Baud300 = 001H ;

Baud1200 = 002H ;

Baud2400 = 003H ;

Baud4800 = 004H ;

Baud9600 = 005H ;

Baud19200 = 006H ;

BaudMIDI = 007H ;

(* PaperType *)

FanFold = 000H ;

Single = 000H ;

(* PrintPitch *)

Pica = 0000H ;

Elite = 0400H ;

Fine = 0800H ;

(* PrintQuality *)

Draft = 0000H ;

Letter = 0100H ;

(* PrintSpacing *)

SixLPI = 0000H ;

EightLPI = 0200H ;

(* Print Image *)

ImagePositive = 000H ;

ImageNegative = 001H ;

(* PrintAspect *)

AspectHoriz = 000H ;

AspectVert = 001H ;

(* PrintShade *)

ShadeBW = 000H ;

ShadeGreyScale = 001H ;

ShadeColor = 002H ;

(* PaperSize *)

USLetter = 000H ;

```
USLegal =      010H ;
NTractor =     020H ;
WTractor =     030H ;
Custom =      040H ;
```

```
(* PrinterType *)
```

```
CustomName =   000H ;
AlphaP101 =    001H ;
Brother15XL =   002H ;
CBMMPS1000 =    003H ;
Diab630 =      004H ;
DiabADVD25 =   005H ;
DiabC150 =     006H ;
Epson =        007H ;
EpsonJX80 =    008H ;
Okimate20 =    009H ;
QumeLP20 =     00AH ;
```

```
PROCEDURE GetDefPrefs ( VAR Buffer : Preferences ; Size : CARDINAL ) : ADDRESS;
(* Get a copy of the default preferneces *)
```

```
PROCEDURE GetPrefs ( VAR Buffer : Preferences ; Size : CARDINAL ) : ADDRESS ;
(* Get current setting of preferneces *)
```

```
(* Intuition internal procedures *)
```

```
PROCEDURE SetPrefs ( VAR Buffer : Preferences ; Size : CARDINAL ;
                    RealThing : BOOLEAN ) ;
(* Set internal state of Intuition preferences *)
```

```
END Preferences.
```

DEFINITION MODULE PrinterDevice;

(**

Commodore Amiga printer device module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

**)-----

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 30-Dec-85

Version : 0.00a 30-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM IO IMPORT CmdNonStd, IORequest;

CONST

PrinterName = "printer.device";

CONST

(* IORequest commands *)

PRDrawWrite = CmdNonStd + 0;

PRDPrtCommand = CmdNonStd + 1;

PRDDumpReport = CmdNonStd + 2;

TYPE

(* printer command definitions sent by PRDPrtCommand *)

PrinterCommand = (aRIS, (* ESCc reset *)
aRIN, (* ESC#1 initialize *)
aIND, (* ESCD line feed *)
aNEL, (* ESCE return, line feed *)
aRI, (* ESCM reverse line feed *)

aSGR0, (* ESC[0m normal character set *)
aSGR3, (* ESC[3m italics on *)
aSGR23, (* ESC[23m italics off *)

aSGR4, (* ESC[4m underline on *)
aSGR24, (* ESC[24m underline off *)
aSGR1, (* ESC[1m boldface on *)
aSGR22, (* ESC[22m boldface off *)
aSFC, (* SGR30-39 set foreground color *)
aSBC, (* SGR40-49 set background color *)

aSHORP0, (* ESC[0w normal pitch *)
aSHORP2, (* ESC[2w elite on *)
aSHORP1, (* ESC[1w elite off *)
aSHORP4, (* ESC[4w condensed fine on *)
aSHORP3, (* ESC[3w condensed off *)
aSHORP6, (* ESC[6w enlarged on *)
aSHORP5, (* ESC[5w enlarged off *)

aDEN6, (* ESC[6"z shadow print on *)
aDEN5, (* ESC[5"z shadow print off *)
aDEN4, (* ESC[4"z doublestrike on *)
aDEN3, (* ESC[3"z doublestrike off *)
aDEN2, (* ESC[2"z near letter quality on *)
aDEN1, (* ESC[1"z near letter quality off *)

aSUS2, (* ESC[2v superscript on *)
aSUS1, (* ESC[1v superscript off *)
aSUS4, (* ESC[4v subscript on *)
aSUS3, (* ESC[3v subscript off *)
aSUS0, (* ESC[0v normalize the line *)
aPLU, (* ESCL partial line up *)
aPLD, (* ESCK partial line down *)

aFNT0, (* ESC(B US char set *)
aFNT1, (* ESC(R French char set *)
aFNT2, (* ESC(K German char set *)
aFNT3, (* ESC(A UK char set *)
aFNT4, (* ESC(E Danish I char *)
aFNT5, (* ESC(H Sweden char *)
aFNT6, (* ESC(Y Italian char set *)
aFNT7, (* ESC(Z Spanish char set *)
aFNT8, (* ESC(J Japanese char set *)
aFNT9, (* ESC(6 Norweign char set *)
aFNT10, (* ESC(C Danish II char set *)

aPROP2, (* ESC[2p proportional on *)

```

aPROP1, (* ESC[1p proportional off *)
aPROP8, (* ESC[8p proportional clear *)
aTSS, (* ESC[n E set proportional offset *)
aJFY5, (* ESC[5 F auto left justify *)
aJFY7, (* ESC[7 F auto right justify *)
aJFY6, (* ESC[6 F auto full justify *)
aJFY8, (* ESC[8 F auto justify off *)
aJFY3, (* ESC[3 F letter space, justify *)
aJFY1, (* ESC[1 F word fill, auto center *)

aVERP8, (* ESC[8z 1/8" line spacing *)
aVERP1, (* ESC[1z 1/6" line spacing *)
aSLPP, (* ESC[nt set form length n *)
aPERF, (* ESC[nq perf skip n (n>0) *)
aPERF8, (* ESC[8q perf skip off *)

aLMS, (* ESC#9 left margin set *)
aRMS, (* ESC#8 right margin set *)
aTMS, (* ESC#8 top margin set *)
aBMS, (* ESC#2 bottom marg set *)
aSTBM, (* ESC[Pn1;Pn2r top and bottom margins *)
aSLRM, (* ESC[Pn1;Pn2s left and right margins *)
aCAM, (* ESC#3 clear margins *)

aHTS, (* ESC#H set horiz tab *)
aVTS, (* ESC#J set vertical tabs *)
aTBC8, (* ESC[8g clear horizontal tab *)
aTBC3, (* ESC[3g clear all horizontal tabs *)
aTBC1, (* ESC[1g clear vertical tabs *)
aTBC4, (* ESC[4g clear all vertical tabs *)
aTBCALL, (* ESC#4 clear all horizontal & vertical tab *)
aTBSALL, (* ESC#5 set default tabs *)
aEXTEND); (* ESC[Pn"x extended commands *)

```

TYPE

(* ask for a text style change *)

IOPrCmdReq = RECORD

ioReq: IORequest;

ioPrCmd: CARDINAL; (* PrinterCommand *)

ioParm0: BYTE; (* parameter 1 *)

ioParm1: BYTE; (* parameter 2 *)

ioParm2: BYTE; (* parameter 3 *)

```
        ioParm3: BYTE; (* parameter 4 *)  
    END;
```

TYPE

(* dump rasterport to the printer *)

IODRPreq = RECORD

```
    ioReq: IORequest;  
    ioRastPort: ADDRESS; (* RastPortPtr, raster port *)  
    ioColorMap: ADDRESS; (* ColorMapPtr, color map *)  
    ioModes: LONGCARD; (* graphics viewport modes *)  
    ioSrcX: CARDINAL; (* source x origin *)  
    ioSrcY: CARDINAL; (* source y origin *)  
    ioSrcWidth: CARDINAL; (* source x width *)  
    ioSrcHeight: CARDINAL; (* source x height *)  
    ioDestCols: LONGINT; (* destination x width *)  
    ioDestRows: LONGINT; (* destination y height *)  
    ioSpecial: LONGCARD; (* option flags *)  
END;
```

END PrinterDevice.

DEFINITION MODULE RandomNumbers;

(** -----

Commodore Amiga pseudo-random number module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 28-Jan-86

Version : 0.00a 28-Jan-86 Paul Curtis, TDI.
Original

*)

PROCEDURE Random(maxvalue: CARDINAL): CARDINAL;

(* generate a pseudo-random number.

maxvalue: the value returned will be in the range 0..maxvalue;

returns: a pseudo-random number. *)

PROCEDURE Seed(seed: LONGCARD);

(* set the seed for the generator. *)

END RandomNumbers.

DEFINITION MODULE Rasters;

(** -----

Commodore Amiga raster area module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 13-Jan-86

Version : 0.00a 13-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;

FROM GraphicsLibrary IMPORT BitMapPtr, DrawingModeSet;

TYPE

RasInfoPtr = POINTER TO RasInfo;

RasInfo = RECORD

next: RasInfoPtr; (* used for dual playfield *)

bitMap: BitMapPtr;

rxOffset: CARDINAL; (* scroll offsets in this BitMap *)

ryOffset: CARDINAL;

END;

TYPE

TmpRasPtr = POINTER TO TmpRas;

TmpRas = RECORD

rasPtr: ADDRESS;

size: LONGINT;

END;

TYPE

RastPortFlags = (FirstDot, (* draw the first dot of this line? *)

OneDot, (* use one dot mode for drawing lines *)

```

    DBuffer, (* flag set when RastPorts are double-buffered *)
    AreaOutline, (* used by areafiller *)
    RPF4,
    NoCrossFill, (* areafills have no crossovers *)
    RPF6, RPF7, RPF8, RPF9, RPF10,
    RPF11, RPF12, RPF13, RPF14, RPF15);
RastPortFlagSet = SET OF RastPortFlags;

```

TYPE

```

RastPortPtr = POINTER TO RastPort;
RastPort = RECORD
    layer: ADDRESS; (* LayerPtr *)
    bitMap: BitMapPtr;
    AreaPtrn: ADDRESS; (* ptr to areafill pattern *)
    tmpRas: TmpRasPtr;
    areaInfo: ADDRESS; (* AreaInfoPtr *)
    gelsInfo: ADDRESS; (* GelsInfoPtr *)
    Mask: BYTE; (* write mask for this raster *)
    FgPen: BYTE; (* foreground pen for this raster *)
    BgPen: BYTE; (* background pen for this raster *)
    AOIPen: BYTE; (* areafill outline pen *)
    DrawMode: DrawingModeSet; (* mode for fill, lines, and text *)
    AreaPtSz: BYTE;
    linpatcnt: BYTE; (* current line drawing pattern preshift *)
    dummy: BYTE;
    Flags: RastPortFlagSet;
    LinePtrn: BITSET; (* 16 bits for textured lines *)
    cpx: INTEGER; (* current pen position *)
    cpy: INTEGER;
    minterms: ARRAY [0..7] OF BYTE;
    PenWidth: INTEGER;
    PenHeight: INTEGER;
    Font: ADDRESS; (* TextFontPtr; current font address *)
    AlgoStyle: BYTE; (* the algorithmically generated style *)
    TxFlags: BYTE; (* text specific flags *)
    TxHeight: CARDINAL; (* text height *)
    TxWidth: CARDINAL; (* text nominal width *)
    TxBaseline: CARDINAL; (* text baseline *)
    TxSpacing: INTEGER; (* text spacing (per character) *)
    RPUser: ADDRESS;
    wordreserved: ARRAY [0..6] OF CARDINAL;

```

```
longreserved: ARRAY [0..1] OF LONGCARD;  
reserved: ARRAY [0..7] OF BYTE; (* for future use *)  
END;
```

```
PROCEDURE AllocRaster(width, height: CARDINAL): ADDRESS;  
(* allocate space for a bit plane.
```

```
width: the width of the bit plane in pixels.  
height: the height of the bit plane in pixels.
```

```
returns: 0 => not enough memory, otherwise pointer to allocated  
region of memory. *)
```

```
PROCEDURE FreeRaster(raster: ADDRESS; width, height: CARDINAL);  
(* release an allocated bit plane to the free memory pool.
```

```
raster: the bit plane that was allocated by AllocRaster.  
width: the width of the raster.  
height: the height of the raster. *)
```

```
PROCEDURE InitRastPort(VAR rp: RastPort);  
(* initialise a raster port.
```

```
rp: the raster port to initialise. *)
```

```
PROCEDURE InitTmpRas(VAR tmpras: TmpRas; buffer: ADDRESS; size: LONGCARD);  
(* initialise a temporary raster for areafill, floodfill, text.
```

```
tmpras: the temporary raster structure to initialise.  
buffer: the buffer to allocate for this raster.  
size: the size of the buffer. *)
```

```
PROCEDURE ScrollRaster(VAR rp: RastPort; dx,dy: INTEGER;  
minx,miny, maxx,maxy: CARDINAL);  
(* scroll raster towards 0,0.
```

```
rp: the rastport to scroll.  
dx,dy: the distance to move towards 0,0.  
minx,miny: the upper left corner of the rectangle to move.  
maxx,maxy: the lower right corner of the rectangle to move. *)
```

```
PROCEDURE SetRast(VAR rp: RastPort; pen: CARDINAL);  
  (* fill a raster port with one colour.  
  
    rp: the raster port to fill.  
    pen: the pen to set the raster to. *)
```

```
END Rasters.
```


DEFINITION MODULE Regions;

(** -----

Commodore Amiga regions module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 30-Dec-85

Version : 0.00a 30-Dec-85 Paul Curtis, TDI.
Original

*)

FROM GraphicsLibrary IMPORT Rectangle;

TYPE

RegionRectanglePtr = POINTER TO RegionRectangle;

RegionRectangle = RECORD

next: RegionRectanglePtr;

prev: RegionRectanglePtr;

bounds: Rectangle;

END;

TYPE

RegionPtr = POINTER TO Region;

Region = RECORD

bounds: Rectangle;

regionRectangle: RegionRectanglePtr;

END;

PROCEDURE AndRectRegion(VAR region: Region; VAR rectangle: Rectangle);

(* perform 2D AND operation of rectangle with region -> region.

region: the region structure for source and result.

rectangle: the rectangle structure. *)

```

PROCEDURE ClearRegion(VAR region: Region);
  (* set the region to size 0.
    region: the region to set to 0. *)

PROCEDURE DisposeRegion(VAR region: Region);
  (* return all space for this region to the free memory pool.
    region: the region to dispose of. *)

PROCEDURE NewRegion(): RegionPtr;
  (* create a region of size 0 and return a pointer to it.
    returns: a pointer to the new region structure. *)

PROCEDURE NotRegion(VAR region: Region);
  (* perform 2D NOT operation of Region -> Region.
    region: the region structure for source and result. *)

PROCEDURE OrRectRegion(VAR region: Region; VAR rectangle: Rectangle);
  (* perform 2D OR operation of rectangle with region -> region.
    region: the region structure for source and result.
    rectangle: the rectangle structure. *)

PROCEDURE XorRectRegion(VAR region: Region; VAR rectangle: Rectangle);
  (* perform 2D XOR operation of rectangle with region -> region.
    region: the region structure for source and result.
    rectangle: the rectangle structure. *)

END Regions.

```

DEFINITION MODULE Requesters ;

(* -----

TDI Modula-2/Amiga : Intuition/Requesters

----- *)

(* ----- *)

(* (c) Copyright TDI Software Ltd. 1986. All Rights Reserved *)

(* ----- *)

FROM SYSTEM IMPORT BYTE ;

FROM Intuition IMPORT Requester, Window, IntuitionText, IDCMPFlagsSet ;

(* NB. The Intuition library must be loaded before calling this module
(see Intuition.def). *)

PROCEDURE AutoRequest (VAR Win : Window ;
 VAR BodyText : IntuitionText ;
 VAR PositiveText : IntuitionText ;
 VAR NegativeText : IntuitionText ;
 PositiveFlags : IDCMPFlagsSet ;
 NegativeFlags : IDCMPFlagsSet ;
 Width, Height : INTEGER) : BOOLEAN ;

(* Automatically build and get response from a requester. *)

PROCEDURE BuildSysRequest (VAR Win : Window ;
 VAR BodyText : IntuitionText ;
 VAR PositiveText : IntuitionText ;
 VAR NegativeText : IntuitionText ;
 Flags : IDCMPFlagsSet ;
 Width, Height : INTEGER) : BOOLEAN ;

(* Build and display a system requester *)

PROCEDURE EndRequest (VAR Req : Requester ; VAR Win : Window) ;

(* End the request and reset the window *)

PROCEDURE FreeSysRequest (VAR Win : Window) ;

(* Free memory used by a call to BuildSysRequest *)

PROCEDURE InitRequester (VAR Req : Requester) ;

(* Initialises a requester structure *)

```
PROCEDURE Request ( VAR Req : Requester ; VAR Win : Window ) : BOOLEAN ;  
(* Activate a requester *)
```

```
END Requesters.
```

DEFINITION MODULE Resident;

(** -----

Commodore Amiga resident module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 29-Jan-86

Version : 0.00a 29-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;

TYPE

ResidentPtr = POINTER TO Resident;

Resident = RECORD

rtMatchWord: CARDINAL; (* word to match on *)
rtMatchTag: ResidentPtr; (* pointer to the above *)
rtEndSkip: ADDRESS; (* address to continue scan *)
rtFlags: BYTE; (* various tag flags *)
rtVersion: BYTE; (* release version number *)
rtType: BYTE; (* type of module *)
rtPri: BYTE; (* initialization priority *)
rtName: ADDRESS; (* pointer to node name *)
rtIdString: ADDRESS; (* pointer to ident string *)
rtInit: PROCEDURE; (* init code *)

END;

CONST

MatchWord = 04AFCH;

CONST

AutoInit = 80H;

ColdStart = 1;

```
PROCEDURE FindResident(VAR name: ARRAY OF CHAR): ADDRESS;  
PROCEDURE InitResident(resident: ADDRESS; segList: ADDRESS);  
  
END Resident.
```

DEFINITION MODULE RoundRobinScheduler;

(** -----

Commodore Amiga simple scheduler module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Unknown, TDI Software, Inc. 09-Jan-86

Version : 0.00b 30-Jan-86 Paul Curtis, TDI.
Renamed a few things for simplicity.
0.00a 09-Jan-86 Unknown, TDI.
Original

*)

PROCEDURE InitialiseScheduler;

(* Initialise the scheduler. Any tasks are disposed *)

PROCEDURE CreateTask(Proc: PROC);

(* Allocate space for a process, and link into ready list *)

PROCEDURE StartSchedulingTasks;

(* Transfers control from program process to the new task *)

PROCEDURE StopSchedulingTasks;

(* Transfers control from the old task back to the program 'process' *)

PROCEDURE NextTask;

(* Change new task to old task, transfer to next new task on list. *)

END RoundRobinScheduler.

DEFINITION MODULE Screens ;

(* -----

TDI Modula-2/Amiga : Intuition/Screens

----- *)

(* ----- *)

(* (c) Copyright TDI Software Ltd. 1986. All Rights Reserved *)

(* ----- *)

FROM SYSTEM IMPORT BYTE, ADDRESS ;

FROM GraphicsLibrary IMPORT BitMapPtr ;

FROM Text IMPORT TextAttrPtr ;

FROM Intuition IMPORT Screen, GadgetPtr, ScreenFlagsSet, ScreenFlags ;

(* NB. The Intuition library must be loaded before calling this module
(see Intuition.def). *)

(* Compound flags *)

CONST

ScreenType = ScreenFlagsSet{ScreenType0..ScreenType3} ;(* all the screens *)

WBenchScreen = ScreenFlagsSet{ScreenType0} ; (* The Workbench *)

CustomScreen = ScreenFlagsSet{ScreenType0..ScreenType3} ; (* for special look*)

TYPE

NewScreen = RECORD

LeftEdge, TopEdge,

Width, Height, Depth : INTEGER ; (* screen dimensions *)

DetailPen, BlockPen : BYTE ; (* bar/border/gadget rendering *)

ViewModes : CARDINAL ; (* Modes for the ViewPort (and View) *)

Type : ScreenFlagsSet ;(* the Screen type (see above) *)

Font : TextAttrPtr ; (* Screen's default text attributes *)

DefaultTitle : ADDRESS ;(* the default title for this Screen *)

Gadgets : GadgetPtr ; (* your own Gadgets for this Screen *)

CustomBitMap : BitMapPtr ;

END ; (* OF RECORD *)

(* Notes :

CustomBitMap: If you are opening a CUSTOMSCREEN and already have a BitMap
that you want used for your Screen, you set the flags CUSTOMBITMAP in the

Types variable and you set this variable to point to your BitMap structure. The structure will be copied into your Screen structure, after which you may discard your own BitMap if you want.

*)

```
PROCEDURE CloseScreen ( VAR Scr : Screen ) ;  
(* Close the screen *)
```

```
PROCEDURE CloseWorkBench () : BOOLEAN ;  
(* Close the workbench screen *)
```

```
PROCEDURE DisplayBeep ( VAR Scr : Screen ) ;  
(* "Beeps" the video display *)
```

```
PROCEDURE MakeScreen ( VAR Scr : Screen ) ;  
(* Do a MakeVPort of a custom screen *)
```

```
PROCEDURE MoveScreen ( VAR Scr : Screen ; DeltaX, DeltaY : INTEGER ) ;  
(* Move the screen *)
```

```
PROCEDURE OpenScreen ( VAR NewScr : NewScreen ) : ADDRESS ;  
(* Open a new screen *)
```

```
PROCEDURE OpenWorkBench () : BOOLEAN ;  
(* Open the workbench screen *)
```

```
PROCEDURE ScreenToBack ( VAR Scr : Screen ) ;  
(* Send screen to back *)
```

```
PROCEDURE ScreenToFront ( VAR Scr : Screen ) ;  
(* Send screen to font *)
```

```
PROCEDURE ShowTitle ( VAR Scr : Screen ; ShowIt : BOOLEAN ) ;  
(* Set the screen title bar display mode *)
```

```
PROCEDURE WBenchToBack () : BOOLEAN ;  
(* Send workbench screen to back *)
```

```
PROCEDURE WBenchToFront () : BOOLEAN ;  
(* Send workbench screen to front *)
```

```
END Screens.
```

DEFINITION MODULE SerialDevice;

(** -----
Commodore Amiga serial device module
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved
----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 30-Dec-85

Version : 0.00a 30-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE;
FROM IO IMPORT IOStdReq, CmdNonStd;

CONST
SerialName = "serial.device";

CONST
(* serial device commands *)
SDCmdQuery = CmdNonStd + 0; (* query line/port status *)
SDCmdBreak = CmdNonStd + 1; (* send break signal *)
SDCmdSetParams = CmdNonStd + 2; (* set serial device parameters *)

TYPE
IOTArray = RECORD
TermArray0: ARRAY [0..3] OF CHAR;
TermArray1: ARRAY [0..3] OF CHAR;
END;

TYPE
SerStatus = (NotBusy,
NotPaperOut,
NotSelect,
NotDataSetReady,

```

        NotClearToSend,
        NotCarrierDetect,
        NotReadyToSend,
        NotDataTerminalReady,
        ReadOverrun,
        BreakSent,
        BreakReceived,
        TransmitXOFFed,
        ReceiveXOFFed,
        Reserved0,
        Reserved1,
        Reserved2);
SerStatusSet = SET OF SerStatus;

```

TYPE

```

SerFlag = (SerParityOn, (* parity is on for receive *)
SerParityOdd, (* parity is odd *)
SerQueuedBrk, (* queue this break IOResult *)
SerHighSpeed, (* high speed mode *)
SerShared, (* non-exclusive access *)
SerEOFMode, (* EOF mode enabled *)
SerDisabled); (* xOn-xOff feature disabled *)
SerFlagSet = SET OF SerFlag;

```

TYPE

```

(* serial device IORB *)
IOExtSer = RECORD
    ioSer: IOStdReq;
    ioCtlChar: ARRAY [0..3] OF CHAR; (* xON,xOFF,INQ,ACK *)
    ioRBufLen: LONGCARD; (* length of serial read buffer, bytes *)
    ioWBufLen: LONGCARD; (* length of serial write buffer, bytes *)
    ioBaud: LONGCARD; (* baud rate requested (true baud) *)
    ioBrkTime: LONGCARD; (* break signal duration in microseconds *)
    ioTermArray: IOTArray; (* termination character array *)
    ioReadLen: BYTE; (* nr. bits per read character *)
    ioWriteLen: BYTE; (* nr. bits per written character *)
    ioStopBits: BYTE; (* nr. stopbits for read *)
    ioSerFlags: SerFlagSet; (* serial device flags *)
    ioStatus: SerStatusSet; (* serial device status *)
END;

```

CONST

(* ioFlags *)

SerActive = 4; (* request is queued, or is current *)

SerAbort = 5; (* request aborted *)

SerQueued = 6; (* request queued *)

SerBufRead = 7; (* from read buffer *)

CONST

TillEOF = 0FFFFFFFFH; (* if IOExtSer.ioLength = TillEOF, transmit all
characters until an EOF byte is reached,
EOF = 0C *)

CONST

(* serial device errors *)

SerErrDevBusy = 1;

SerErrBaudMismatch = 2;

SerErrInvBaud = 3;

SerErrBufErr = 4;

SerErrInvParam = 5;

SerErrLineErr = 6;

SerErrNotOpen = 7;

SerErrPortReset = 8;

SerErrParityErr = 9;

SerErrInitErr = 10;

SerErrTimerErr = 11;

SerErrBufOverflow = 12;

END SerialDevice.

DEFINITION MODULE Sprites;

(** -----

Commodore Amiga sprite module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 30-Dec-85

Version : 0.00a 30-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;

FROM Views IMPORT ViewPort;

CONST

AnySprite = -1; (* indicated to GetSprite that any sprite will do *)

TYPE

SimpleSprite = RECORD

posCtlData: ADDRESS;

height: CARDINAL; (* the sprite's height *)

x: CARDINAL; (* current position *)

y: CARDINAL;

num: CARDINAL; (* sprite number *)

END;

PROCEDURE ChangeSprite(VAR vp: ViewPort; VAR s: SimpleSprite; image: ADDRESS);

(* change the sprite image pointer.

vp: the view port the sprite is relative to.

s: the simple sprite to change.

image: pointer to data for new sprite. *)

```

PROCEDURE FreeSprite(num: CARDINAL);
  (* return sprite for use by others.

    num: the sprite number to return. *)

PROCEDURE GetSprite(VAR ss: SimpleSprite; num: INTEGER): INTEGER;
  (* attempt to get a sprite for the simple sprite manager.

    ss: the simple sprite structure.
    num: -1 => get any available sprite, 0..7 => get this sprite.

    returns: -1 => no sprites available, otherwise the sprite
              that was allocated. *)

PROCEDURE MoveSprite(VAR vp: ViewPort; VAR ss: SimpleSprite; x,y: CARDINAL);
  (* move sprite to a point relative to the top left of viewport.

    vp: the view port for this sprite.
    sprite: the simple sprite.
    x,y: the position relative to the to left of the view port. *)

END Sprites.

```

DEFINITION MODULE Storage;

(** -----

Commodore Amiga standard storage module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 28-Dec-85

Version : 0.00a 28-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;

PROCEDURE ALLOCATE(VAR addr: ADDRESS; amount: CARDINAL);

(* allocate a portion of the heap - HALT if no memory available.

addr: returns where the memory portion was allocated, word aligned.
amount: the number of bytes to allocate, will be a word multiple. *)

PROCEDURE DEALLOCATE(VAR addr: ADDRESS; amount: CARDINAL);

(* return a portion of memory to the heap.

addr: the portion of memory returned by ALLOCATE.
amount: the size of the block given to ALLOCATE. *)

PROCEDURE CreateHeap(amount: LONGCARD): BOOLEAN;

(* create a heap - must be called before ALLOCATE/DEALLOCATE or NEW/DISPOSE.

amount: the number of bytes to set aside for the heap.

returns: TRUE => heap created OK, otherwise no memory available
for heap. *)

PROCEDURE HeapLeft(): LONGCARD;

(* return amount of storage left on the heap.

returns: total amount of storage left on the heap. Note that
an object of this size may not be able to be created
with ALLOCATE as the free storage left may be
fragmented. *)

PROCEDURE DestroyHeap;

(* return the region allocated for the heap to the system. *)

END Storage.

DEFINITION MODULE Streams;

(** -----

Commodore Amiga standard streams module

(c) Copyright 1985, 1985 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 12-Dec-85

Version : 0.00a 12-Dec-85 Paul Curtis, TDI.
Original based upon ETH standard.

*)

(* This is a Medos based standard stream module. As stated in "Programming in Modula-2" (p. 189, third edition), this module is allowed to export file system dependent representations for the STREAM type.

Note: This module will open the DOS library if it is not already open. *)

FROM SYSTEM IMPORT WORD;
FROM DOSFiles IMPORT FileHandle;

CONST Done = 0; (* STREAM.res = Done => operation successful;
otherwise the AmigaDOS error code. *)

TYPE

STREAM = RECORD

handle: FileHandle; (* Filehandle returned by DOS *)

eof: BOOLEAN; (* TRUE => end of file reached *)

res: INTEGER; (* result field, returned by DOS *)

endpos: LONGCARD; (* last position in file *)

END;

PROCEDURE Connect(VAR s: STREAM; f: FileHandle);

(* connect stream s with open file f.

f is a filehandle returned by DOS *)

```

PROCEDURE Disconnect(VAR s: STREAM);
  (* terminate association of s with f. DOS is left to close f. *)

PROCEDURE WriteWord(s: STREAM; w: WORD);
  (* Write a word to the stream. *)

PROCEDURE WriteChar(s: STREAM; c: CHAR);
  (* Write a character to the stream. *)

PROCEDURE ReadWord(s: STREAM; VAR w: WORD);
  (* Read a word from the stream. *)

PROCEDURE ReadChar(s: STREAM; VAR c: CHAR);
  (* Read a character from the stream. *)

PROCEDURE WriteString(s: STREAM; VAR str: ARRAY OF CHAR);
  (* Write a string to the stream. *)

PROCEDURE ReadString(s: STREAM; VAR str: ARRAY OF CHAR);
  (* Write a string to the stream. *)

PROCEDURE Reset(s: STREAM);
  (* Position stream to beginning of file. *)

PROCEDURE SetPos(s: STREAM; pos: LONGCARD);
  (* Set file cursor of s, pos bytes from beginning of file. *)

PROCEDURE GetPos(s: STREAM; VAR pos: LONGCARD);
  (* Return current file cursor. *)

```

END Streams.

DEFINITION MODULE String;

(* String utility module *)

(**-----**)

(* (c) Copyright 1984 32DOS Ltd All Rights Reserved *)

(**-----**)

(* (c) Copyright 1985 TDI Ltd All Rights Reserved *)

(**-----**)

EXPORT QUALIFIED

(* CONST *) MaxChars,

(* TYPE *) Strings, CompareResults,

(* PROC *) InitStringModule, Assign, Insert, Delete, Copy,
Concat, Length, Compare, Pos, GetTerminator,
SetTerminator;

CONST MaxChars = 80;

TYPE Strings = ARRAY[0..MaxChars] OF CHAR;

(* Base string type though it is possible to declare strings
of any length *)

CompareResults = (Greater, Equal, Less);

(* Result of a comparison of two strings. *)

PROCEDURE InitStringModule;

(* Initialises the string module *)

PROCEDURE Assign((* TO *) VAR Dest : ARRAY OF CHAR;

VAR Source : ARRAY OF CHAR);

(* Equivalent to Dest := Source *)

PROCEDURE Insert((* String *) VAR SubStr : ARRAY OF CHAR;

(* Into *) VAR Str : ARRAY OF CHAR;

(* At point *) Index : CARDINAL);

(* starting at point Index in string Str inserts SubStr *)

```

PROCEDURE Delete((% From      %) VAR Str   : ARRAY OF CHAR;
                  (% starting at %) Index : CARDINAL;
                  (% No of chars %) Len   : CARDINAL   );

```

(% removes Len characters from Str starting at point Index %)

```

PROCEDURE Copy((% origin      %) VAR Str   : ARRAY OF CHAR;
                (% from point %) Index   : CARDINAL;
                (% No of chars %) Len     : CARDINAL;
                (% into        %) VAR Result : ARRAY OF CHAR);

```

(% Copies Len characters starting at Index in Str into Result %)

```

PROCEDURE Concat(          VAR S1,
                  (% With  %) S2   : ARRAY OF CHAR;
                  (% giving %) VAR Result : ARRAY OF CHAR);

```

(% equivalent to Result := S1 + S2 %)

```

PROCEDURE Length(VAR Str : ARRAY OF CHAR) : CARDINAL;

```

(% function giving the length of the passed string %)

```

PROCEDURE Compare( VAR S1 : ARRAY OF CHAR;
                   VAR S2 : ARRAY OF CHAR) : CompareResults;

```

(% Procedure to compare two strings.

Result is : Equal - if strings are same length with same
content (possibly empty);

Greater - if identical upto the end of one (possibly
empty) string but other is longer;

Less - if on char by char compare one string has a char
less than the char at the same position in the
other string (in ASCII code value). *)

```

PROCEDURE Pos ( VAR Source : ARRAY OF CHAR;
                VAR Match  : ARRAY OF CHAR;

```

```
        Start : CARDINAL;  
        VAR Where : CARDINAL ) : BOOLEAN;
```

```
(* Finds position of Match in Source. Returns TRUE if a match found *)
```

```
PROCEDURE SetTerminator(Ch : CHAR);
```

```
(* Sets the string terminator character to Ch *)
```

```
PROCEDURE GetTerminator() : CHAR;
```

```
(* Returns the value of the string terminator character *)
```

```
END (* OF MODULE *) String.
```

DEFINITION MODULE Tasks;

```
(** -----  
Commodore Amiga Tasks module  
(c) Copyright 1985, 1985 TDI Software, Inc. All Rights Reserved  
----- **)
```

(* VERSION FOR COMMODORE AMIGA

```
Original Author : Paul Curtis, TDI Software, Inc. 04-Dec-85  
Version        : 0.00a 04-Dec-85 Paul Curtis, TDI.  
Original
```

*)

```
FROM SYSTEM IMPORT ADDRESS, BYTE;  
FROM Modes IMPORT Mode;  
FROM Lists IMPORT List;
```

CONST

```
MyTask = 0; (* Passed to FindTask or RenTask to indicate current task *)  
GeneralFinalPC = 0; (* Passed to addTask for general task finalisation *)  
AnySignal = -1; (* Passed to AllocSignal to indicate any signal will do *)  
AnyTrap = -1; (* Passed to AllocTrap to indicate any trap will do *)  
NoSignals = -1; (* Returned by AllocSignal when no signals available *)  
NoTraps = -1; (* Returned by AllocTrap when no trap available *)  
MaxTrapNr = 15;  
MaxSignalNr = 31;
```

TYPE

```
(* these definitions allow -1 as AllocTrap & AllocSignal may return -1  
to indicate that the allocation failed *)
```

```
SIGNAL = [NoSignals..MaxSignalNr];  
TRAP = [NoTraps..MaxTrapNr];
```

```
SignalSet = SET OF [0..MaxSignalNr];  
TrapSet = SET OF [0..MaxTrapNr];
```

```
FlagBits = (TBProcTime, TBU1, TBU2, TBU3, TBStackCheck, TBExcept,
            TBSwitch, TBLaunch);
```

```
TaskState = (TSInvalid, TSAdded, TSTrun, TSReady, TSWait, TSExcept,
            TSRemoved);
```

CONST

```
(* Predefined signals *)
```

```
SigAbort = 0;
```

```
SigChild = 1;
```

```
SigBlit = 4;
```

```
SigDOS = 7;
```

TYPE

```
TaskPtr = POINTER TO Task;
```

```
Task = RECORD
```

```
    tcNode: Node;
```

```
    tcFlags: SET OF FlagBits;
```

```
    tcState: SET OF TaskState;
```

```
    tcIDNestCnt: BYTE; (* Intr disabled nesting *)
```

```
    tcTDNestCnt: BYTE; (* Task disabled nesting *)
```

```
    tcSigAlloc: SignalSet; (* signals allocated *)
```

```
    tcSigWait: SignalSet; (* awaited signals *)
```

```
    tcSigRecvd: SignalSet; (* received signals *)
```

```
    tcSigExcept: SignalSet; (* signals that give exceptions *)
```

```
    tcTrapAlloc: TrapSet; (* traps allocated *)
```

```
    tcTrapAble: TrapSet; (* traps enabled *)
```

```
    tcExceptData: ADDRESS; (* points to exception data *)
```

```
    tcExceptCode: PROCEDURE; (* exception code *)
```

```
    tcTrapData: ADDRESS; (* points to trap data *)
```

```
    tcTrapCode: PROCEDURE; (* trap code *)
```

```
    tcSPReg: ADDRESS; (* stack pointer *)
```

```
    tcSPLower: ADDRESS; (* stack lower bound *)
```

```
    tcSPUpper: ADDRESS; (* stack upper bound + 2 *)
```

```
    tcSwitch: PROCEDURE; (* task losing CPU *)
```

```
    tcLaunch: PROCEDURE; (* task getting CPU *)
```

```
    tcMemEntry: List; (* allocated memory *)
```

```
    tcUserData: ADDRESS; (* per task *)
```

```
END;
```

```
PROCEDURE AddTask(task: TaskPtr; initialPC, finalPC: ADDRESS);
```

```
(* add a task to the system.
```

task: pointer to the task control block.
initialPC: the initial entry point into the task.
finalPC: the finalisation address. If zero, a general system
finalisation routine will be called. *)

PROCEDURE AllocSignal(signalNum: SIGNAL): SIGNAL;

(* Allocate a signal bit - do not allocate while in TRAP processing code.

signalNum: the desired signal number, or -1 indicating any free
signal will do.

returns: -1 => signal cannot be allocated, otherwise the allocated
signal number. *)

PROCEDURE AllocTrap(trapNum: TRAP): TRAP;

(* allocate a processor trap vector.

trapNum: the desired trap number, or -1 indicating any free trap
vector will do.

returns: -1 => trap cannot be allocated, otherwise the allocated
trap number. *)

PROCEDURE FindTask(name: ADDRESS): TaskPtr;

(* find a task with a given name, or find onself.

name: 0 => return current task pointer, otherwise the name
of the task to find, null terminated.

returns: 0 => task not found, otherwise a pointer to the task
control block. *)

PROCEDURE FreeSignal(signalNum: SIGNAL);

(* free a signal bit - do not free while in TRAP processing code.

signalNum: the signal number to free in this task. *)

PROCEDURE FreeTrap(trapNum: TRAP);

(* free a processor trap.

trapNum: the trap number to free. *)

PROCEDURE RenTask(task: TaskPtr);

(* remove a task from the system - no resource deallocation performed.

task: 0 => remove self, otherwise pointer to task node to be removed. *)

PROCEDURE SetExcept(newSignals, signalMask: SignalSet): SignalSet;

(* define certain signals to cause exceptions.

newSignals: the new values for the signals specified in signalMask.

signalMask: the set of signals to be affected.

returns: the old exception signals for this task.

e.g. Get current signals that cause exceptions:

SetExcept(SignalSet{},SignalSet{})

Define signal 3 to cause exception, all others unaffected:

SetExcept(SignalSet{3},SignalSet{3})

Define signal 2 to cause exception, 13 not to cause exception:

SetExcept(SignalSet{2,13},SignalSet{2})

Define that no signals cause exceptions:

SetExcept(SignalSet{0..31},SignalSet{}) *)

PROCEDURE SetSignal(newSignals, signalMask: SignalSet): SignalSet;

(* define the state of this tasks signals.

newSignals: the new values for the signals specified in signalMask.

signalMask: the set of signals to be affected.

returns: the old signals for this task.

See SetExcept for details on how newSignals and signalMask interact. *)

PROCEDURE SetTaskPri(task: TaskPtr; pri: CARDINAL): CARDINAL;

(* get and set the priority of a task - may cause a context switch.

task: the task to change the priority of.

pri: the new priority, 0..511.

returns: old task priority. *)

PROCEDURE Signal(task: TaskPtr; signals: SignalSet);

(* signal a task.

task: the task to send the signals to.

signals: the signals that will be sent to the task. This may or may not awaken the task, as the task may be waiting for different signals. *)

PROCEDURE Wait(signalSet: SignalSet): SignalSet;

(* Wait for one or more signals - do not call in supervisor mode.

signalSet: signals that will awaken the sleeping task. Any one of the signals within this set will allow the sleeping task to become ready again.

returns: the signals that caused the task to wake. *)

END Tasks.

DEFINITION MODULE Terminal;

(** -----

Commodore Amiga standard terminal module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 08-Jan-86

Version : 0.00a 08-Jan-86 Paul Curtis, TDI.
Original

*)

(* Note: This module tries to open the DOS library for access to the
Input and Output files; this modifies the variable DOSBase in
module DOS. Therefore, if your application uses this module you
do not need to open the DOS library with an OpenLibrary call,
but you do need to close it with a CloseLibrary(DOSBase). *)

PROCEDURE Read (VAR c: CHAR);

(* Read a character from the keyboard, if one is not available then wait. *)

PROCEDURE BusyRead(VAR c: CHAR);

(* Read a character from the keyboard, if one is not available then return
0C *)

PROCEDURE ReadAgain;

(* Causes the last character read to be returned again upon the next call
to Read or BusyRead *)

PROCEDURE Write(c: CHAR);

(* Write the character to the screen *)

PROCEDURE WriteLn;

(* Go to a new line *)

PROCEDURE WriteString(VAR s: ARRAY OF CHAR);

(* Write the string to the screen *)

END Terminal.

DEFINITION MODULE Text;

(** -----

Commodore Amiga graphic text module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 13-Jan-86

Version : 0.00a 13-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM GraphicsLibrary IMPORT BitMap;
FROM Rasters IMPORT RastPort;
FROM Ports IMPORT Message;

TYPE

FontStyles = (Underlined, (* underlined under baseline *)
Bold, (* thickened by a pixel *)
Italic, (* slanted right *)
Extended); (* extended face, wider than normal *)
FontStyleSet = SET OF FontStyles;

CONST

NormalStyle = FontStyleSet{}; (* normal text *)

TYPE

FontFlags = (ROMFont, (* font is in ROM *)
DiskFont, (* font from diskfont.library *)
RevPath, (* designed path is reversed (e.g. left) *)
FF3, FF4,
Proportional, (* character sizes can vary from nominal *)
Designed, (* size is "designed", not constructed *)
Removed); (* the font has been removed *)

FontFlagSet = SET OF FontFlags;

TYPE

TextAttrPtr = POINTER TO TextAttr;

TextAttr = RECORD

taName: ADDRESS; (* a BPTR *)

taYSIZE: CARDINAL; (* height of the font *)

taStyle: FontStyleSet; (* intrinsic font style *)

taFlags: FontFlagSet; (* font preferences and flags *)

END;

TYPE

TextFontPtr = POINTER TO TextFont;

TextFont = RECORD

tfMessage: Message; (* reply message for font removal *)

tfYSIZE: CARDINAL; (* font height *)

tfStyle: FontStyleSet; (* font style *)

tfFlags: FontFlagSet; (* preferences and flags *)

tfXSIZE: CARDINAL; (* nominal font width *)

tfBaseline: CARDINAL; (* distance from the top to baseline *)

tfBoldSmear: CARDINAL; (* smear to affect a bold *)

tfAccessors: CARDINAL; (* access count *)

tfLoChar: BYTE; (* first character described here *)

tfHiChar: BYTE; (* last character described here *)

tfCharData: ADDRESS; (* bit character data *)

tfModulo: CARDINAL; (* row modulo for strike font data *)

tfCharLoc: ADDRESS; (* ptr to location data for strike font *)

tfCharSpace: ADDRESS; (* ptr to words of prop. spacing data *)

tfCharKern: ADDRESS; (* ptr to words of kerning data *)

END;

PROCEDURE BitBitMap(VAR srcBM: BitMap; srcX, srcY: CARDINAL;

VAR dstBM: BitMap; dstX, dstY: CARDINAL;

sizeX, sizeY: CARDINAL;

minTerm: CARDINAL;

mask: BITSET;

tempA: ADDRESS): CARDINAL;

(* move a rectangle in a raster.

srcBM: the source bitmap.

srcX, srcY: the upper left corner of the source within the bitmap.
dstBM: the destination bitmap.
dstX, dstY: the upper left corner of the destination within the bitmap.
sizeX, sizeY: the dimensions of the rectangle to copy.
minterm: the logic function to apply to the rectangle.
mask: the write mask to apply to the blit.
tempA: buffer for one source line.

returns: number of planes involved in the blit. *)

```
PROCEDURE BltTemplate(source: ADDRESS; srcX: CARDINAL; srcMod: CARDINAL;  
                     VAR destRastPort: RastPort; destX, destY: CARDINAL;  
                     sizeX, sizeY: CARDINAL);
```

(* cookie cut a shape in the rectangle to the rastport.

source: the template.
srcX: the x position of the template.
srcMod: the source modulo.
destRastPort: the rastport to put the shape in.
destX, destY: the upper left corner in the destination for the shape.
sizeX, sizeY: the size of the shape. *)

```
PROCEDURE ClearEOL(VAR rp: RastPort);
```

(* clear from current position to end of line.

rp: the rastport to clear to eol in. *)

```
PROCEDURE ClearScreen(VAR rp: RastPort);
```

(* clear from current position to end of rastport.

rp: the rastport to clear to eos in. *)

```
PROCEDURE TextLength(VAR rp: RastPort; VAR str: ARRAY OF CHAR;  
                    count: CARDINAL): CARDINAL;
```

(* determine raster length of text data.

rp: the rastport where the text attributes reside.
str: the text data.
count: the number of characters in the string.

returns: the number of pixels in the X direction that the text

would occupy. *)

PROCEDURE Text(VAR rp: RastPort; VAR str: ARRAY OF CHAR; count: CARDINAL);
(* write text characters.

rp: the rastport where the text will be written.
str: the text data to write.
count: the number of characters in the string. *)

PROCEDURE AskSoftStyle(VAR rp: RastPort): FontStyleSet;
(* get the soft style bits of the current font.

rp: the rastport to get the text attributes of.
returns: the set of styles that are algorithmically generated. *)

PROCEDURE SetSoftStyle(VAR rp: RastPort;
style, enable: FontStyleSet): FontStyleSet;
(* set the soft style of a font.

rp: the rastport to set the soft style of.
style: the new style to set, subject to enable.
enable: the style bits to be changed.
returns: the new style for the font, according to restrictions. *)

PROCEDURE OpenFont(VAR textAttr: TextAttr): TextFontPtr;
(* get a pointer to a system font.

textAttr: the text attributes to look for.
returns: 0 => font could not be found, otherwise a pointer
to the open system font. *)

PROCEDURE CloseFont(VAR font: TextFont);
(* release a pointer to the system font.

font: the font to close. *)

PROCEDURE SetFont(VAR rp: RastPort; VAR font: TextFont);
(* set the text font and attributes for a rastport.

rp: the rastport receiving the new font style.
font: the font style to set for the rastport. *)

PROCEDURE AskFont(VAR rp: RastPort; VAR textAttr: TextAttr);

(* get the text attributes of a rastport.

rp: the rastport to get the attributes of.
textAttr: the resulting text attributes of the rastports font. *)

PROCEDURE AddFont(VAR textFont: TextFont);

(* add a font to the system list.

textFont: the font to add to the system list. *)

PROCEDURE RemFont(VAR textFont: TextFont): LONGINT;

(* remove a font from the system list.

textFont: the font to remove from the system list.

returns: 0 => font removed OK, otherwise another process is
still using this font, and font not removed. *)

END Text.

DEFINITION MODULE TimerDevice;

(** -----

Commodore Amiga timer module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 22-Dec-85

Version : 0.00a 22-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;

FROM IO IMPORT IORequest, CmdNonStd;

CONST

UnitMicroHz = 0; (* timer is driven by 8520 programmable timer *)

UnitVBlank = 1; (* timer is driven by vertical blank interrupt *)

(* UnitMicroHz: precision = 2 micro seconds, but drifts as system
load increases; typically accurate to within 5%.
UnitVBlank: resolution = 16667 microseconds, but very stable. It
is very cheap to use; should be used for waiting
for time intervals > 0.5 seconds. *)

CONST

TimerName = "timer.device";

TYPE

TimeVal = RECORD

tvSecs: LONGCARD;

tvMicro: LONGCARD;

END;

TYPE

```

TimerRequest = RECORD
    trNode: IORequest;
    trTime: TimeVal;
END;

CONST
    (* IOCommands used for adding a timer *)
    TRAddRequest = CmdNonStd; (* add request to time time *)
    TRGetSysTime = CmdNonStd + 1; (* get the system time *)
    TRSetSysTime = CmdNonStd + 2; (* set the system time *)

CONST
    (* values returned by CmpTime *)
    TimeLess    = -1;
    TimeSame    = 0;
    TimeGreater = +1;

PROCEDURE AddTime(VAR dest, source: TimeVal);
    (* add one TimeVal to another.

    dest: the TimeVal where the result will be stored.
    source: the TimeVal to add to dest (dest := dest + source). *)

PROCEDURE CmpTime(VAR tv1, tv2: TimeVal): CARDINAL;
    (* compare two TimeVal structures.

    tv1: the first TimeVal.
    tv2: the second TimeVal.

    returns : -1 => tv1 < tv2,
               0 => tv1 = tv2,
               +1 => tv1 > tv2. *)

PROCEDURE SubTime(VAR dest, source: TimeVal);
    (* subtract one TimeVal from another.

    dest: the TimeVal where the result will be stored.
    source: the TimeVal to subtract from dest (dest := dest - source). *)

END TimerDevice.

```

DEFINITION MODULE TrackDiskDevice;

(** -----

Commodore Amiga disk tracking module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 30-Dec-85

Version : 0.00a 30-Dec-85 Paul Curtis, TDI.
Original

*)

FROM IO IMPORT CmdNonStd, CmdWrite, CmdRead, CmdUpdate, CmdClear, IOStdReq;

CONST

TrackDiskName = "trackdisk.device";

CONST

(* track disk device commands *)

TDMotor = CmdNonStd + 0; (* control the disk's motor *)
TDSeek = CmdNonStd + 1; (* explicit seek, for testing *)
TDFormat = CmdNonStd + 2; (* format disk *)
TDRRemove = CmdNonStd + 3; (* notify when disk changes *)
TDChangeNum = CmdNonStd + 4; (* nr. disk changes *)
TDChangeState = CmdNonStd + 5; (* is there a disk in the drive? *)
TDProtStatus = CmdNonStd + 6; (* is the disk write protected? *)

CONST

(* track disk device errors *)

TDErrNotSpecified = 20;
TDErrNoSecHdr = 21;
TDErrBadSecPreamble = 22;
TDErrBadSecID = 23;
TDErrBadHdrSum = 24;
TDErrBadSecSum = 25;
TDErrTooFewSecs = 26;

```

TDErrBadSecHdr      = 27;
TDErrWriteProt      = 28;
TDErrDiskChanged    = 29;
TDErrSeekError      = 30;
TDErrNoMem          = 31;
TDErrBadUnitNum     = 32;
TDErrBadDriveType   = 33;
TDErrDriveInUse     = 34;

```

TYPE

```

(* track disk I/O request *)
IOExtTD = RECORD
    iotdReq: IOStdReq;
    iotdCount: LONGCARD;
    iotdSecLabel: LONGCARD;
END;

```

CONST

```

(* physical drive characteristics *)
NrCylinders = 80; (* nr. cylinders on a disk *)
MaxCylinders = 100; (* nr. cyls to look for during calibration *)
NrSecs = 11; (* nr. sectors on a track *)
NrHeads = 2; (* nr. heads on a drive *)
MaxRetry = 10; (* maximum number of retries before giving up *)
NrTracks = NrCylinders * NrHeads; (* nr. tracks on a disk *)
NrUnits = 4; (* up to four physical disk drives *)

```

CONST

```

(* sizes before MFM encoding *)
TDSector = 512; (* bytes held on one sector *)
TDSecShift = 9;

```

CONST

```

TDExtCom = 8000H; (* internal use only *)

```

CONST

```

(* extended commands *)
ETDWrite = TDExtCom + CmdWrite;
ETDRead = TDExtCom + CmdRead;
ETDMotor = TDExtCom + TDMotor;
ETDSeek = TDExtCom + TDSeek;
ETDFormat = TDExtCom + TDFormat;
ETDUpdate = TDExtCom + CmdUpdate;

```

```
ETDClear = TExtCom + CmdClear;
```

```
CONST
```

```
LabelSize = 16; (* labels are 16 bytes per sector *)
```

```
END TrackDiskDevice.
```

DEFINITION MODULE TranslatorLibrary;

(** -----

Commodore Amiga translator module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 27-Dec-85

Version : 0.00a 27-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;

CONST

TranslatorName = "translator.library";

VAR

TranslatorBase: ADDRESS;

CONST

TRNotUsed = -1;

TRNoMem = -2; (* can't allocate memory *)

TRMakeBad = -4; (* error in MakeLibrary call *)

PROCEDURE Translate(VAR inString: ARRAY OF CHAR; inLen: LONGCARD;

VAR outBuf: ARRAY OF CHAR; outLen: LONGCARD): LONGINT;

(* convert an Englist string into phonetics.

inString: the English string to translate.

inLen: the length of the English string.

outBuf: the array to hold the phonetic codes.

outLen: the length of the output array.

returns: 0 => no error, otherwise an -ve number that is a pointer into

inString where the translation was broken off. *)

END TranslatorLibrary.

DEFINITION MODULE Trapper;

(** -----

Commodore Amiga Modula-2 error trapping module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 04-Dec-85

Version : 0.00a 04-Dec-85 Paul Curtis, TDI.
Original

*)

(* If this module is imported by any program, full run-time diagnostics
are given for any Modula-2 or system error.

*)

END Trapper.

DEFINITION MODULE Views;

(** -----
Commodore Amiga view module
(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved
----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 30-Dec-85

Version : 0.00a 30-Dec-85 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM Copper IMPORT CopListPtr, cprlistptr, UCopListPtr;
FROM Rasters IMPORT RasInfoPtr;

TYPE
Modes = (u1, GenlockVideo, Lace, u8, u10, u20, PFBA, ExtraHalfBright,
GenlockAudio, u200, DualPF, HAM, u1000, VPHide, Sprites, Hires);
ModeSet = SET OF Modes;

TYPE
ViewPortPtr = POINTER TO ViewPort;
ViewPort = RECORD
next: ViewPortPtr;
colorMap: ADDRESS; (* ColorMapPtr *)
(* color map for this view port,
0 => MakeVPort assumes defaults *)
dspIns: CopListPtr; (* used by MakeVPort() *)
sprIns: CopListPtr; (* used by sprite *)
clrIns: CopListPtr; (* used by sprite *)
uCopIns: UCopListPtr; (* user copper list *)
dWidth: CARDINAL;
dHeight: CARDINAL;
dxOffset: CARDINAL;

```

dyOffset: CARDINAL;
modes: ModeSet;
reserved: CARDINAL;
rasInfo: RasInfoPtr;
END;

```

TYPE

```

ViewPtr = POINTER TO View;
View = RECORD
    viewport: ViewPortPtr;
    LOFCprList: cprlistptr; (* long frame copper list *)
    SHFCprList: cprlistptr; (* short frame copper list *)
    dyOffset: CARDINAL;
    dxOffset: CARDINAL;
    modes: ModeSet;
END;

```

```

PROCEDURE FreeVPortCopLists(VAR vp: ViewPort);
(* deallocate all intermediate copper lists & headers from the viewport.

vp: the viewport to deallocate copper lists from. *)

```

```

PROCEDURE InitView(VAR view: View);
(* initialise a view structure.

view: the view to initialise to default. *)

```

```

PROCEDURE InitVPort(VAR vp: ViewPort);
(* initialise a viewport structure.

vp: the viewport to initialise to default. *)

```

```

PROCEDURE LoadView(VAR view: View);
(* use a copper list to create a new display.

view: the view structure to show on the display. *)

```

```

PROCEDURE MakeVPort(VAR view: View; VAR vp: ViewPort);
(* generate a display copper list.

view: the view into which the vp instructions are placed.

```

vp: the viewport to merge into view. *)

PROCEDURE MrgCop(VAR view: View);

(* merge together copper instructions.

view: the view structure to create the copper list from. *)

PROCEDURE ScrollVPort(VAR vp: ViewPort);

(* scroll the viewport by dx,dy towards 0,0.

vp: the viewport to scroll. *)

PROCEDURE WaitBOVP(VAR vp: ViewPort);

(* wait for beam to reach bottom of viewport.

vp: the viewport to wait for beam to pass. *)

PROCEDURE WaitTOF;

(* wait for the top of the next video frame. *)

END Views.

DEFINITION MODULE Windows ;

(* -----
TDI Modula-2/Amiga : Intuition/Windows
----- *)

(* ----- *)
(* (c) Copyright TDI Software Ltd. 1986. All Rights Reserved *)
(* ----- *)

FROM SYSTEM IMPORT ADDRESS ;
FROM Intuition IMPORT Window, WindowFlags, WindowFlagsSet, NewWindow,
IDCMPFlagsSet, IntuiMessagePtr, Requester ;
(* NB. The Intuition library must be loaded before calling this module
(see Intuition.def). *)

CONST

(* Compound window flags *)

RefreshBits = WindowFlagsSet{Refresh0, Refresh1} ;
SmartRefresh = WindowFlagsSet{} ;
SimpleRefresh = WindowFlagsSet{Refresh0} ;
SuperBitMap = WindowFlagsSet{Refresh1} ;
OtherRefresh = WindowFlagsSet{Refresh0, Refresh1} ;

SuperUnused = WindowFlagsSet{Wi18, Wi19, Wi22, Wi31} ; (* bits of Flag unused *)

PROCEDURE BeginRefresh (VAR Win : Window) ;
(* Setup a window for optimised refreshing *)

PROCEDURE ClearDMRequest (VAR Win : Window) : BOOLEAN ;
(* Clears the DMRequest of the window *)

PROCEDURE ClearPointer (VAR Win : Window) ;
(* Clears the pointer definition from a window *)

PROCEDURE CloseWindow (VAR Win : Window) ;
(* Closes a window *)

PROCEDURE EndRefresh (VAR Win : Window ; Complete : BOOLEAN) ;

(* Ends the optimised refresh state of the window *)

PROCEDURE ModifyIDCMP (VAR Win : Window ; IDCMPFlags : IDCMPFlagsSet) ;
(* Modify the state of the windows IDCMP *)

PROCEDURE MoveWindow (VAR Win : Window ; DeltaX, DeltaY : LONGINT) ;
(* Ask Intuition to move a window *)

PROCEDURE OpenWindow (VAR NewWin : NewWindow) : ADDRESS ;
(* Open new window. Returns pointer to window *)

PROCEDURE ReportMouse (VAR Win : Window ; On : BOOLEAN) ;
(* Set state of mouse reporting for this window *)

PROCEDURE SetDMRequest (VAR Win : Window ; VAR DMR : Requester) : BOOLEAN ;
(* Set the DMRequest of a window *)

PROCEDURE SetPointer (VAR Win : Window ; Pointer : ADDRESS ;
 Height, Width : CARDINAL ; Xoffset, Yoffset : INTEGER) ;
(* Set window with its own pointer *)

PROCEDURE SetWindowTitles (VAR Win : Window ;
 VAR WindowTitle, ScreenTitle : ARRAY OF CHAR) ;
(* Set the windows title for both window and screen *)

PROCEDURE SizeWindow (VAR Win : Window ; DeltaX, DeltaY : INTEGER) ;
(* Ask Intuition to size a window *)

PROCEDURE ViewAddress () : ADDRESS ;
(* Returns the address of the Intuition view structure *)

PROCEDURE ViewPortAddress (VAR Win : Window) : ADDRESS ;
(* Returns the address of a window's ViewPort structure *)

PROCEDURE WindowLimits (VAR Win : Window ;
 MinWidth, MinHeight, MaxWidth, MaxHeight : CARDINAL)
 : BOOLEAN ;
(* Set the limits of a window *)

PROCEDURE WindowToBack (VAR Win : Window) ;
(* Send the window to the back *)

PROCEDURE WindowToFront (VAR Win : Window) ;
(* Send the window to the front *)

END Windows.

DEFINITION MODULE Workbench;

(** -----

Commodore Amiga Workbench module

(c) Copyright 1985, 1986 TDI Software, Inc. All Rights Reserved

----- **)

(* VERSION FOR COMMODORE AMIGA

Original Author : Paul Curtis, TDI Software, Inc. 24-Jan-86

Version : 8.00a 24-Jan-86 Paul Curtis, TDI.
Original

*)

FROM SYSTEM IMPORT ADDRESS;

FROM Nodes IMPORT Node;

FROM Lists IMPORT List;

FROM Ports IMPORT Message, MsgPortPtr;

FROM Intuition IMPORT Gadget, NewWindow, Image, PropInfo, WindowPtr;

TYPE

WBArgPtr = POINTER TO WBArg;

WBArg = RECORD

 waLock: ADDRESS; (* a lock descriptor, BPTR *)

 waName: ADDRESS; (* a string relative to that lock *)

END;

TYPE

WBStartup = RECORD

 smMessage: Message; (* a standard message structure *)

 smProcess: MsgPortPtr; (* the process descriptor for us *)

 smSegment: ADDRESS; (* a descriptor for your code, BPTR *)

 smNumArgs: LONGINT; (* nr. elements in ArgList *)

 smToolWindow: ADDRESS; (* description of window *)

 smArgList: WBArgPtr; (* the arguments themselves *)

END;

TYPE

WBOBJECTType = (WBDisk, WBDrawer, WBTool, WBProject, WBGarbage, WBDevice,
WBKick);

TYPE

WBOBJECTPtr = POINTER TO WBOBJECT;

TYPE

(* only ddNewWindow, ddCurrentX, ddCurrentY written to disk. *)

DrawerDataPtr = POINTER TO DrawerData;

DrawerData = RECORD

ddNewWindow: NewWindow; (* args to open window *)
ddCurrentX: LONGINT; (* current x coordinate of origin *)
ddCurrentY: LONGINT; (* current y coordinate of origin *)
ddMinX: LONGINT; (* smallest x coordinate in window *)
ddMinY: LONGINT; (* smallest y coordinate in window *)
ddMaxX: LONGINT; (* largest x coordinate in window *)
ddMaxY: LONGINT; (* latgest y coordinate in window *)
ddHorizScroll: Gadget;
ddVertScroll: Gadget;
ddUpMove: Gadget;
ddDownMove: Gadget;
ddLeftMove: Gadget;
ddRightMove: Gadget;
ddHorizImage: Image;
ddVertImage: Image;
ddHorizProp: PropInfo;
ddVertProp: PropInfo;
ddDrawerWin: WindowPtr; (* pointer to drawers window *)
ddObject: WBOBJECTPtr; (* back pointer to drawer object *)
ddChildren: List; (* where our children hang out *)
ddLock: LONGINT;

END;

TYPE

DiskObject = RECORD

doMagic: CARDINAL; (* magic number at start of file *)
doVersion: CARDINAL; (* version number *)
doGadget: Gadget; (* copy of in-core gadget *)
doType: WBOBJECTType;
doDefaultTool: POINTER TO CHAR;

```

doToolTypes: POINTER TO POINTER TO CHAR;
doCurrentX: LONGINT;
doCurrentY: LONGINT;
doDrawerData: DrawerDataPtr;
doToolWindow: POINTER TO CHAR; (* only applies to tools *)
doStackSize: LONGINT; (** only applies to tools *)
END;

```

CONST

```

DiskMagicNr = 0E310H;
DiskVersion = 1; (* current version number *)

```

TYPE

```

FreeList = RECORD
    flNumFree: CARDINAL;
    flMemList: List;
END;

```

TYPE

```

WBOBJECT = RECORD
    woMasterNode: Node; (* all objects are on this list *)
    woSiblings: Node; (* list of drawer members *)
    woSelectNode: Node; (* list of all selected objects *)
    woUtilityNode: Node; (* function specific linkages *)
    woParent: WBOBJECTPtr;

    (* specified as BITSET; test bit 0 for condition. *)
    woIconDisp: BITSET; (* icon is currently in a window *)
    woDrawerOpen: BITSET; (* we're a drawer, and it is open *)
    woSelected: BITSET; (* our icon is selected *)
    woBackground: BITSET; (* set if icon is in background *)

    woType: WBOBJECTType; (* what flavor object is this? *)
    woUseCount: CARDINAL; (* number of references to this obj *)
    woName: ADDRESS; (* this object's textual name *)
    woNameXOffset: CARDINAL; (* where to put the name *)
    woNameYOffset: CARDINAL;

    woDefaultTool: ADDRESS;
    woDrawerData: DrawerDataPtr; (* if this is a drawer or disk *)
    woIconWin: WindowPtr; (* each object's icon lives here *)
    woCurrentX: LONGINT; (* virtual X in drawer *)

```

```

woCurrentY: LONGINT; (* virtual Y in drawer *)
woToolTypes: POINTER TO ADDRESS; (* the types for this tool *)
woGadget: Gadget;
woFreeList: FreeList; (* this objects free list *)
woToolWindow: ADDRESS; (* character string for tool's window *)
woStackSize: LONGCARD; (* how much stack to give to this *)
woLock: LONGINT; (* if this tool is in the backdrop *)

```

END;

TYPE

```

WBPortMsgType = (None,
                  PStd, (* a "standard Potion" message *)
                  ToolExit, (* exit message from tools *)
                  DiskChange, (* DOS telling us of a disk change *)
                  Timer, (* timer tick *)
                  Closedown, (* unimplemented *)
                  IOProc); (* unimplemented *)

```

TYPE

```

GadgetIDSpecial = (WrkBObject, (* normal workbench object *)
                   HorizScroll, (* the horiz scroll gadget for a drawer *)
                   VertScroll, (* the vert scroll gadget for a drawer *)
                   LeftScroll, (* move one window left *)
                   RightScroll, (* move one window right *)
                   UpScroll, (* move one window up *)
                   DownScroll, (* move one window down *)
                   ObjName); (* the name field for an object *)

```

CONST

```
BackFill = 1;
```

CONST

```
NoIconPosition = 80000000H;
```

END Workbench.

TDI Modula-2/Amiga definition items cross reference :

ADDRESS8	Gels	TYPE
ADKBit	ADKBits	TYPE
ADKBitSet	ADKBits	TYPE
ALLOCATE	Storage	PROCEDURE
AUserStuff	Gels	TYPE
AbleICR	CIAResource	PROCEDURE
AbortIO	IO	PROCEDURE
ActivationFlags	Intuition	TYPE
ActivationFlagsSet	Intuition	TYPE
AddAnimOb	Gels	PROCEDURE
AddBob	Gels	PROCEDURE
AddDevice	Devices	PROCEDURE
AddFont	Text	PROCEDURE
AddFreeList	IconLibrary	PROCEDURE
AddGadget	Gadgets	PROCEDURE
AddHead	Lists	PROCEDURE
AddICRVector	CIAResource	PROCEDURE
AddIntServer	Interrupts	PROCEDURE
AddLibrary	Libraries	PROCEDURE
AddPort	Ports	PROCEDURE
AddTail	Lists	PROCEDURE
AddTask	Tasks	PROCEDURE
AddTime	TimerDevice	PROCEDURE
AddVSprite	Gels	PROCEDURE
Alert	Exec	PROCEDURE
AllocCList	CListLibrary	PROCEDURE
AllocEntry	Memory	PROCEDURE
AllocMem	Memory	PROCEDURE
AllocRaster	Rasters	PROCEDURE
AllocRemember	Intuition	PROCEDURE
AllocSignal	Tasks	PROCEDURE
AllocTrap	Tasks	PROCEDURE
AllocWBObject	IconLibrary	PROCEDURE
Allocate	Memory	PROCEDURE
AlohaWorkbench	Intuition	PROCEDURE

AndRectRegion	Regions	PROCEDURE
AnimComp	Gels	TYPE
AnimCompPtr	Gels	TYPE
AnimOb	Gels	TYPE
AnimObPtr	Gels	TYPE
Animate	Gels	PROCEDURE
AreaDraw	Areas	PROCEDURE
AreaEnd	Areas	PROCEDURE
AreaInfo	Areas	TYPE
AreaInfoPtr	Areas	TYPE
AreaMove	Areas	PROCEDURE
AskFont	Text	PROCEDURE
AskSoftStyle	Text	PROCEDURE
Assign	String	PROCEDURE
AudioChannelSet	AudioDevice	TYPE
AudioChannels	AudioDevice	TYPE
AutoRequest	Requesters	PROCEDURE
AvailFont	DiskFontLibrary	TYPE
AvailFonts	DiskFontLibrary	PROCEDURE
AvailFontsHeader	DiskFontLibrary	TYPE
AvailFontsHeaderPtr	DiskFontLibrary	TYPE
AvailMem	Memory	PROCEDURE
BPTR	DOSLibrary	TYPE
BPTRFromPtr	AmigaUtils	PROCEDURE
BSTR	DOSLibrary	TYPE
BUserStuff	Gels	TYPE
Base	ConvToString	TYPE
BeginIO	IO	PROCEDURE
BeginRefresh	Windows	PROCEDURE
BitMap	GraphicsLibrary	TYPE
BitMapPtr	GraphicsLibrary	TYPE
BlitBitMap	Text	PROCEDURE
BlitClear	GraphicsLibrary	PROCEDURE

BltPattern	Blitter	PROCEDURE
BltTemplate	Text	PROCEDURE
Bob	Gels	TYPE
BobFlagSet	Gels	TYPE
BobFlags	Gels	TYPE
BobPtr	Gels	TYPE
Border	Intuition	TYPE
BorderPtr	Intuition	TYPE
BoundaryOff	PenUtils	PROCEDURE
BplCon0	BlitterHardware	TYPE
BplCon0Set	BlitterHardware	TYPE
BplCon1	BlitterHardware	TYPE
BplCon1Set	BlitterHardware	TYPE
BuildSysRequest	Requesters	PROCEDURE
BusyRead	Terminal	PROCEDURE
CARDINAL8	Gels	TYPE
CASEX	AMIGAX	PROCEDURE
CBump	Copper	PROCEDURE
CEND	CopperUtils	PROCEDURE
CIAA	CIAHardware	VAR
CIAB	CIAHardware	VAR
CIABase	CIAResource	VAR
CIACR	CIAHardware	TYPE
CIACRBits	CIAHardware	TYPE
CIAICR	CIAHardware	TYPE
CIAICRBits	CIAHardware	TYPE
CIAType	CIAHardware	TYPE
CINIT	CopperUtils	PROCEDURE
CLStrings	CommandLine	TYPE
CLineLen	AMIGAX	VAR
CLinePtr	AMIGAX	VAR
CListBase	CListLibrary	VAR
CMOVE	CopperUtils	PROCEDURE
CMove	Copper	PROCEDURE

CWAIT	CopperUtils	PROCEDURE
CWait	Copper	PROCEDURE
Cause	Interrupts	PROCEDURE
ChangeSprite	Sprites	PROCEDURE
CharCases	ASCII	TYPE
CharIsASCII	ASCII	PROCEDURE
CharIsControl	ASCII	PROCEDURE
CharIsPrintable	ASCII	PROCEDURE
CheckIO	IO	PROCEDURE
ClearDMRequest	Windows	PROCEDURE
ClearEOL	Text	PROCEDURE
ClearMenuStrip	Menus	PROCEDURE
ClearPointer	Windows	PROCEDURE
ClearRegion	Regions	PROCEDURE
ClearScreen	Text	PROCEDURE
ClipANSI	ClipboardDevice	TYPE
ClipBitMap	ClipboardDevice	TYPE
ClipBlit	Layers	PROCEDURE
ClipItem	ClipboardDevice	TYPE
ClipRect	Layers	TYPE
ClipRectPtr	Layers	TYPE
ClipStream	ClipboardDevice	TYPE
ClipboardUnitPartial	ClipboardDevice	TYPE
Close	DOSFiles	PROCEDURE
CloseDevice	Devices	PROCEDURE
CloseFont	Text	PROCEDURE
CloseInput	InOut	PROCEDURE
CloseLibrary	Libraries	PROCEDURE
CloseOutput	InOut	PROCEDURE
CloseScreen	Screens	PROCEDURE
CloseWindow	Windows	PROCEDURE
CloseWorkBench	Screens	PROCEDURE
CmpTime	TimerDevice	PROCEDURE
ColorMap	Colors	TYPE
ColorMapPtr	Colors	TYPE
ColorTable	Colors	TYPE
ColorTablePtr	Colors	TYPE
CommandLineInterface	DOSExtensions	TYPE

CommandLineInterfacePtr	DOSExtensions	TYPE
Compare	String	PROCEDURE
CompareResults	String	TYPE
Concat	String	PROCEDURE
ConcatCList	CListLibrary	PROCEDURE
Connect	Streams	PROCEDURE
CopIns	Copper	TYPE
CopInsPtr	Copper	TYPE
CopList	Copper	TYPE
CopListPtr	Copper	TYPE
CopperListInit	Copper	PROCEDURE
Copy	String	PROCEDURE
CopyCList	CListLibrary	PROCEDURE
CopySBitMap	Layers	PROCEDURE
CreateDir	DOSFiles	PROCEDURE
CreateHeap	Storage	PROCEDURE
CreatePort	PortUtils	PROCEDURE
CreateProc	DOSProcessHandler	PROCEDURE
CreateTask	RoundRobinScheduler	PROCEDURE
CurrentDir	DOSFiles	PROCEDURE
CurrentTime	Intuition	PROCEDURE
Custom	CustomHardware	VAR
CustomType	CustomHardware	TYPE
DBufPacket	Gels	TYPE
DBufPacketPtr	Gels	TYPE
DEALLOCATE	Storage	PROCEDURE
DIVS32	AMIGAX	PROCEDURE
DIVU32	AMIGAX	PROCEDURE
DOSBase	DOSLibrary	VAR
DateStamp	DOSProcessHandler	PROCEDURE
DateStampRec	DOSLibrary	TYPE
Deallocate	Memory	PROCEDURE
Debug	Exec	PROCEDURE
DegToRad	MathLib0	PROCEDURE
Delay	DOSProcessHandler	PROCEDURE

Delete	String	PROCEDURE
DeleteFile	DOSFiles	PROCEDURE
DeletePort	PortUtils	PROCEDURE
DestroyHeap	Storage	PROCEDURE
Device	Devices	TYPE
DeviceList	DOSExtensions	TYPE
DeviceProc	DOSProcessHandler	PROCEDURE
DevicePtr	Devices	TYPE
Disable	Interrupts	PROCEDURE
Disconnect	Streams	PROCEDURE
DiskFontBase	DiskFontLibrary	VAR
DiskFontHeader	DiskFontLibrary	TYPE
DiskFontHeaderPtr	DiskFontLibrary	TYPE
DiskObject	Workbench	TYPE
DisownBlitter	Blitter	PROCEDURE
DisplayAlert	Intuition	PROCEDURE
DisplayBeep	Screens	PROCEDURE
DisplayOff	DMAUtils	PROCEDURE
DisplayOn	DMAUtils	PROCEDURE
DisposeRegion	Regions	PROCEDURE
DoCollision	Gels	PROCEDURE
DoIO	IO	PROCEDURE
Done	InOut	VAR
DosBaseRec	DOSExtensions	TYPE
DosInfo	DOSExtensions	TYPE
DosPacket	DOSExtensions	TYPE
DosPacketPtr	DOSExtensions	TYPE
DoubleClick	Intuition	PROCEDURE
Draw	Pens	PROCEDURE
DrawBorder	Intuition	PROCEDURE
DrawGList	Gels	PROCEDURE
DrawImage	Intuition	PROCEDURE
DrawerData	Workbench	TYPE
DrawerDataPtr	Workbench	TYPE
DrawingModeSet	GraphicsLibrary	TYPE
DrawingModes	GraphicsLibrary	TYPE
DupLock	DOSFiles	PROCEDURE

Enable	Interrupts	PROCEDURE
EndRefresh	Windows	PROCEDURE
EndRequest	Requesters	PROCEDURE
Enqueue	Lists	PROCEDURE
ErrorCause	AMIGAX	TYPE
ErrorContext	AMIGAX	VAR
ErrorContextType	AMIGAX	TYPE
ErrorProcessor	AMIGAX	VAR
ErrorProcessorType	AMIGAX	TYPE
ErrorType	AMIGAX	TYPE
ExNext	DOSFiles	PROCEDURE
Examine	DOSFiles	PROCEDURE
ExecBase	AMIGAX	VAR
ExecBasePtr	ExecBase	TYPE
Execute	DOSCodeLoader	PROCEDURE
Exit	DOSProcessHandler	PROCEDURE
ExitM2	AMIGAX	VAR
FADD	AMIGAX	PROCEDURE
FCMP	AMIGAX	PROCEDURE
FDIV	AMIGAX	PROCEDURE
FLOATX	AMIGAX	PROCEDURE
FMUL	AMIGAX	PROCEDURE
FSUB	AMIGAX	PROCEDURE
FTST	AMIGAX	PROCEDURE
FileHandle	DOSFiles	TYPE
FileHandleBlock	DOSExtensions	TYPE
FileInfoBlock	DOSFiles	TYPE
FileLock	DOSFiles	TYPE
FileLockBlock	DOSExtensions	TYPE
FindName	Lists	PROCEDURE
FindPort	Ports	PROCEDURE
FindResident	Resident	PROCEDURE
FindTask	Tasks	PROCEDURE

FlagBits	Tasks	TYPE
Flood	Pens	PROCEDURE
FlushCList	CListLibrary	PROCEDURE
FontContents	DiskFontLibrary	TYPE
FontContentsHeader	DiskFontLibrary	TYPE
FontContentsHeaderPtr	DiskFontLibrary	TYPE
FontFlagSet	Text	TYPE
FontFlags	Text	TYPE
FontStyleSet	Text	TYPE
FontStyles	Text	TYPE
Forbid	Interrupts	PROCEDURE
FreeCList	CListLibrary	PROCEDURE
FreeColorMap	Colors	PROCEDURE
FreeCopList	Copper	PROCEDURE
FreeCprList	Copper	PROCEDURE
FreeEntry	Memory	PROCEDURE
FreeFreeList	IconLibrary	PROCEDURE
FreeList	Workbench	TYPE
FreeMem	Memory	PROCEDURE
FreeRaster	Rasters	PROCEDURE
FreeRemember	Intuition	PROCEDURE
FreeSignal	Tasks	PROCEDURE
FreeSprite	Sprites	PROCEDURE
FreeSysRequest	Requesters	PROCEDURE
FreeTrap	Tasks	PROCEDURE
FreeVPortCopLists	Views	PROCEDURE
FreeWBOject	IconLibrary	PROCEDURE
FreqRange	NarratorDevice	TYPE
Gadget	Intuition	TYPE
GadgetFlags	Intuition	TYPE
GadgetFlagsSet	Intuition	TYPE
GadgetIDSpecial	Workbench	TYPE
GadgetPtr	Intuition	TYPE
GamePortTrigger	GamePortDevice	TYPE
GelsInfo	Gels	TYPE
GelsInfoPtr	Gels	TYPE
GetCL	CommandLine	PROCEDURE
GetCLBuf	CListLibrary	PROCEDURE

GetCLChar	CListLibrary	PROCEDURE
GetCLWord	CListLibrary	PROCEDURE
GetColorMap	Colors	PROCEDURE
GetDefPrefs	Preferences	PROCEDURE
GetGBuffers	Gels	PROCEDURE
GetIcon	IconLibrary	PROCEDURE
GetMsg	Ports	PROCEDURE
GetPos	Streams	PROCEDURE
GetPrefs	Preferences	PROCEDURE
GetRGB4	Colors	PROCEDURE
GetSprite	Sprites	PROCEDURE
GetTerminator	String	PROCEDURE
GetWBOject	IconLibrary	PROCEDURE
GfxBasePtr	GraphicsBase	TYPE
GraphicsBase	GraphicsLibrary	VAR
HALTX	AMIGAX	PROCEDURE
HeapLeft	Storage	PROCEDURE
IDCMPFlags	Intuition	TYPE
IDCMPFlagsSet	Intuition	TYPE
IEClass	InputEvents	TYPE
IEQualifier	InputEvents	TYPE
IEQualifierSet	InputEvents	TYPE
IOAudio	AudioDevice	TYPE
IOClipReq	ClipboardDevice	TYPE
IODRPRReq	PrinterDevice	TYPE
IOExtPar	ParallelDevice	TYPE
IOExtSer	SerialDevice	TYPE
IOExtTD	TrackDiskDevice	TYPE
IOPArray	ParallelDevice	TYPE
IOPrntCmdReq	PrinterDevice	TYPE
IORequest	IO	TYPE
IORequestPtr	IO	TYPE
IOStdReq	IO	TYPE
IOTArray	SerialDevice	TYPE
IOTTRANSFER	AMIGAX	PROCEDURE

IconBase	IconLibrary	VAR
Image	Intuition	TYPE
ImagePtr	Intuition	TYPE
IncrCLMark	CListLibrary	PROCEDURE
Info	DOSFiles	PROCEDURE
InfoData	DOSFiles	TYPE
InitArea	Areas	PROCEDURE
InitBitMap	GraphicsLibrary	PROCEDURE
InitCLPool	CListLibrary	PROCEDURE
InitGMasks	Gels	PROCEDURE
InitGels	Gels	PROCEDURE
InitMasks	Gels	PROCEDURE
InitRastPort	Rasters	PROCEDURE
InitRequester	Requesters	PROCEDURE
InitResident	Resident	PROCEDURE
InitStringModule	String	PROCEDURE
InitTmpRas	Rasters	PROCEDURE
InitVPort	Views	PROCEDURE
InitView	Views	PROCEDURE
InitialiseScheduler	RoundRobinScheduler	PROCEDURE
Input	DOSFiles	PROCEDURE
InputEvent	InputEvents	TYPE
InputEventPtr	InputEvents	TYPE
Insert	String	PROCEDURE
Insert	Lists	PROCEDURE
IntBit	IntBits	TYPE
IntBitSet	IntBits	TYPE
IntVector	Interrupts	TYPE
IntVectorPtr	Interrupts	TYPE
Interrupt	Interrupts	TYPE
InterruptPtr	Interrupts	TYPE
IntuiMessage	Intuition	TYPE
IntuiMessagePtr	Intuition	TYPE
IntuiTextLength	Intuition	PROCEDURE
Intuition	Intuition	PROCEDURE
IntuitionBase	Intuition	VAR
IntuitionText	Intuition	TYPE
IntuitionTextPtr	Intuition	TYPE
IoErr	DOSFiles	PROCEDURE

IsInteractive	DOSFiles	PROCEDURE
ItemAddress	Menus	PROCEDURE
ItemFlags	Intuition	TYPE
ItemFlagsSet	Intuition	TYPE
KeyMap	ConsoleDevice	TYPE
KeyQualifier	ConsoleDevice	TYPE
KeyQualifierSet	ConsoleDevice	TYPE
LFADD	AMIGAX	PROCEDURE
LFCMP	AMIGAX	PROCEDURE
LFDIU	AMIGAX	PROCEDURE
LFLOATX	AMIGAX	PROCEDURE
LFMUL	AMIGAX	PROCEDURE
LFSUB	AMIGAX	PROCEDURE
LFTST	AMIGAX	PROCEDURE
LTRUNCX	AMIGAX	PROCEDURE
Layer	Layers	TYPE
LayerInfo	Layers	TYPE
LayerInfoPtr	Layers	TYPE
LayerPtr	Layers	TYPE
Length	String	PROCEDURE
LibStateSet	Libraries	TYPE
LibStates	Libraries	TYPE
Library	Libraries	TYPE
LibraryPtr	Libraries	TYPE
List	Lists	TYPE
ListPtr	Lists	TYPE
LoadRGB4	Colors	PROCEDURE
LoadSeg	DOSCodeLoader	PROCEDURE
LoadView	Views	PROCEDURE
Lock	DOSFiles	PROCEDURE
LockLayerRom	Layers	PROCEDURE
MULS32	AMIGAX	PROCEDURE
MULU32	AMIGAX	PROCEDURE

MakeLibrary	Libraries	PROCEDURE
MakeScreen	Screens	PROCEDURE
MakeVPort	Views	PROCEDURE
MarkCList	CListLibrary	PROCEDURE
MemChunk	Memory	TYPE
MemChunkPtr	Memory	TYPE
MemEntry	Memory	TYPE
MemEntryPtr	Memory	TYPE
MemHeader	Memory	TYPE
MemHeaderPtr	Memory	TYPE
MemListHeader	Memory	TYPE
MemListHeaderPtr	Memory	TYPE
MemReqSet	Memory	TYPE
Menu	Intuition	TYPE
MenuFlags	Intuition	TYPE
MenuFlagsSet	Intuition	TYPE
MenuItem	Intuition	TYPE
MenuItemPtr	Intuition	TYPE
MenuPtr	Intuition	TYPE
Message	Ports	TYPE
MessagePtr	Ports	TYPE
ModeSet	Views	TYPE
Modes	Views	TYPE
ModifyIDCMP	Windows	PROCEDURE
ModifyProp	Gadgets	PROCEDURE
MouthRB	NarratorDevice	TYPE
Move	Pens	PROCEDURE
MoveScreen	Screens	PROCEDURE
MoveSprite	Sprites	PROCEDURE
MoveWindow	Windows	PROCEDURE
MrgCop	Views	PROCEDURE
MsgPort	Ports	TYPE
MsgPortPtr	Ports	TYPE
MsgType	Ports	TYPE
NEWPROCESS	AMIGAX	PROCEDURE
NULL	AmigaUtils	PROCEDURE

NarratorRB	NarratorDevice	TYPE
NewList	Lists	PROCEDURE
NewRegion	Regions	PROCEDURE
NewScreen	Screens	TYPE
NewWindow	Intuition	TYPE
NextTask	RoundRobinScheduler	PROCEDURE
Node	Nodes	TYPE
NodePtr	Nodes	TYPE
NodeType	Nodes	TYPE
NotRegion	Regions	PROCEDURE
NumberToString	ConvToString	PROCEDURE
OffGadget	Gadgets	PROCEDURE
OffMenu	Menus	PROCEDURE
OnGadget	Gadgets	PROCEDURE
Open	DOSFiles	PROCEDURE
OpenDevice	Devices	PROCEDURE
OpenDiskFont	DiskFontLibrary	PROCEDURE
OpenFont	Text	PROCEDURE
OpenInput	InOut	PROCEDURE
OpenInputFile	InOut	PROCEDURE
OpenLibrary	Libraries	PROCEDURE
OpenOutput	InOut	PROCEDURE
OpenOutputFile	InOut	PROCEDURE
OpenScreen	Screens	PROCEDURE
OpenWindow	Windows	PROCEDURE
OpenWorkBench	Screens	PROCEDURE
OrRectRegion	Regions	PROCEDURE
Output	DOSFiles	PROCEDURE
OwnBlitter	Blitter	PROCEDURE
Pad	CIAHardware	TYPE
ParFlagSet	ParallelDevice	TYPE
ParStatus	ParallelDevice	TYPE
ParStatusSet	ParallelDevice	TYPE

ParentDir	DOSFiles	PROCEDURE
PeekCLMark	CListLibrary	PROCEDURE
Permit	Interrupts	PROCEDURE
PitchMode	NarratorDevice	TYPE
PitchRange	NarratorDevice	TYPE
PlanePtr	GraphicsLibrary	TYPE
PolyDraw	Pens	PROCEDURE
Pos	String	PROCEDURE
Preferences	Preferences	TYPE
PrintIText	Intuition	PROCEDURE
PrinterCommand	PrinterDevice	TYPE
Process	DOSExtensions	TYPE
ProcessID	DOSProcessHandler	TYPE
ProcessPtr	DOSExtensions	TYPE
PropFlags	Intuition	TYPE
PropFlagsSet	Intuition	TYPE
PropInfo	Intuition	TYPE
ProtMask	DOSFiles	TYPE
PtrFromBPTR	AmigaUtils	PROCEDURE
PutCLBuf	CListLibrary	PROCEDURE
PutCLChar	CListLibrary	PROCEDURE
PutCLWord	CListLibrary	PROCEDURE
PutIcon	IconLibrary	PROCEDURE
PutMsg	Ports	PROCEDURE
PutWBObject	IconLibrary	PROCEDURE
QBSBlit	Blitter	PROCEDURE
QBlit	Blitter	PROCEDURE
RadToDeg	MathLib0	PROCEDURE
Random	RandomNumbers	PROCEDURE
RasInfo	Rasters	TYPE
RasInfoPtr	Rasters	TYPE
RastPort	Rasters	TYPE
RastPortFlagSet	Rasters	TYPE

RastPortFlags	Rasters	TYPE
RastPortPtr	Rasters	TYPE
RateRange	NarratorDevice	TYPE
Read	Terminal	PROCEDURE
Read	InOut	PROCEDURE
Read	DOSFiles	PROCEDURE
ReadAgain	Terminal	PROCEDURE
ReadCard	InOut	PROCEDURE
ReadChar	Streams	PROCEDURE
ReadInt	InOut	PROCEDURE
ReadPixel	Pens	PROCEDURE
ReadString	InOut	PROCEDURE
ReadString	Streams	PROCEDURE
ReadWord	Streams	PROCEDURE
RectFill	Pens	PROCEDURE
Rectangle	GraphicsLibrary	TYPE
RectanglePtr	GraphicsLibrary	TYPE
RefreshGadgets	Gadgets	PROCEDURE
Region	Regions	TYPE
RegionPtr	Regions	TYPE
RegionRectangle	Regions	TYPE
RegionRectanglePtr	Regions	TYPE
RemDevice	Devices	PROCEDURE
RemFont	Text	PROCEDURE
RemHead	Lists	PROCEDURE
RemIBob	Gels	PROCEDURE
RemICRVector	CIAResource	PROCEDURE
RemIntServer	Interrupts	PROCEDURE
RemLibrary	Libraries	PROCEDURE
RemPort	Ports	PROCEDURE
RemTail	Lists	PROCEDURE
RemTask	Tasks	PROCEDURE
RemVSprite	Gels	PROCEDURE
RemakeDisplay	Intuition	PROCEDURE
Remember	Intuition	TYPE
RememberPtr	Intuition	TYPE
Remove	Lists	PROCEDURE
RemoveGadget	Gadgets	PROCEDURE
Rename	DOSFiles	PROCEDURE
ReplyMsg	Ports	PROCEDURE
ReportMouse	Windows	PROCEDURE

Request	Requesters	PROCEDURE
Requester	Intuition	TYPE
RequesterFlags	Intuition	TYPE
RequesterFlagsSet	Intuition	TYPE
RequesterPtr	Intuition	TYPE
Reset	Streams	PROCEDURE
Resident	Resident	TYPE
ResidentPtr	Resident	TYPE
RethinkDisplay	Intuition	PROCEDURE
RootNode	DOSExtensions	TYPE
SHORTSET	CIAHardware	TYPE
SIGNAL	Tasks	TYPE
STACKTEST	AMIGAX	PROCEDURE
STREAM	Streams	TYPE
SYSCALL	AMIGAX	PROCEDURE
SatisfyMsg	ClipboardDevice	TYPE
Screen	Intuition	TYPE
ScreenFlags	Intuition	TYPE
ScreenFlagsSet	Intuition	TYPE
ScreenPtr	Intuition	TYPE
ScreenToBack	Screens	PROCEDURE
ScreenToFront	Screens	PROCEDURE
ScrollRaster	Rasters	PROCEDURE
ScrollVPort	Views	PROCEDURE
Seed	RandomNumbers	PROCEDURE
Seek	DOSFiles	PROCEDURE
SendIO	IO	PROCEDURE
SerFlag	SerialDevice	TYPE
SerFlagSet	SerialDevice	TYPE
SerStatus	SerialDevice	TYPE
SerStatusSet	SerialDevice	TYPE
SetAPen	Pens	PROCEDURE
SetAfPat	PenUtils	PROCEDURE
SetBPen	Pens	PROCEDURE
SetCollision	Gels	PROCEDURE

SetComment	DOSFiles	PROCEDURE
SetDMRequest	Windows	PROCEDURE
SetDrMd	Pens	PROCEDURE
SetDrPt	PenUtils	PROCEDURE
SetExcept	Tasks	PROCEDURE
SetFont	Text	PROCEDURE
SetFunction	Libraries	PROCEDURE
SetICR	CIAResource	PROCEDURE
SetIntVector	Interrupts	PROCEDURE
SetMenuStrip	Menus	PROCEDURE
SetOPen	PenUtils	PROCEDURE
SetPointer	Windows	PROCEDURE
SetPos	Streams	PROCEDURE
SetPrefs	Preferences	PROCEDURE
SetProtection	DOSFiles	PROCEDURE
SetRGB4	Colors	PROCEDURE
SetRast	Rasters	PROCEDURE
SetSR	Interrupts	PROCEDURE
SetSignal	Tasks	PROCEDURE
SetSoftStyle	Text	PROCEDURE
SetTaskPri	Tasks	PROCEDURE
SetTerminator	String	PROCEDURE
SetWindowTitles	Windows	PROCEDURE
SetWrMsk	PenUtils	PROCEDURE
Sex	NarratorDevice	TYPE
ShowTitle	Screens	PROCEDURE
Signal	Tasks	PROCEDURE
SignalSet	Tasks	TYPE
SimpleSprite	Sprites	TYPE
SizeCList	CListLibrary	PROCEDURE
SizeWindow	Windows	PROCEDURE
SoftIntList	Interrupts	TYPE
SortGList	Gels	PROCEDURE
SplitCList	CListLibrary	PROCEDURE
SpriteOff	DMAUtils	PROCEDURE
SpriteOn	DMAUtils	PROCEDURE
StackPtr	AMIGAX	VAR
StackSize	AMIGAX	VAR

StandardPacket	DOSExtensions	TYPE
StartSchedulingTasks	RoundRobinScheduler	PROCEDURE
StopSchedulingTasks	RoundRobinScheduler	PROCEDURE
StringInfo	Intuition	TYPE
Strings	String	TYPE
SubCList	CListLibrary	PROCEDURE
SubTime	TimerDevice	PROCEDURE
SumLibrary	Libraries	PROCEDURE
SuperState	Interrupts	PROCEDURE
SyncSBitMap	Layers	PROCEDURE
TRANSFER	AMIGAX	PROCEDURE
TRAP	Tasks	TYPE
TRUNCX	AMIGAX	PROCEDURE
Task	Tasks	TYPE
TaskPtr	Tasks	TYPE
TaskState	Tasks	TYPE
Text	Text	PROCEDURE
TextAttr	Text	TYPE
TextAttrPtr	Text	TYPE
TextFont	Text	TYPE
TextFontPtr	Text	TYPE
TextLength	Text	PROCEDURE
TimeVal	TimerDevice	TYPE
TimerRequest	TimerDevice	TYPE
TmpRas	Rasters	TYPE
TmpRasPtr	Rasters	TYPE
Translate	TranslatorLibrary	PROCEDURE
TranslatorBase	TranslatorLibrary	VAR
TrapSet	Tasks	TYPE
UCopList	Copper	TYPE
UCopListPtr	Copper	TYPE
UnGetCLChar	CListLibrary	PROCEDURE
UnGetCLWord	CListLibrary	PROCEDURE

UnPutCLChar	CListLibrary	PROCEDURE
UnPutCLWord	CListLibrary	PROCEDURE
Unit	Devices	TYPE
UnitPtr	Devices	TYPE
UnitStateSet	Devices	TYPE
UnitStates	Devices	TYPE
UnloadSeg	DOSCodeLoader	PROCEDURE
Unlock	DOSFiles	PROCEDURE
UnlockLayerRom	Layers	PROCEDURE
UserState	Interrupts	PROCEDURE
VBeamPos	Blitter	PROCEDURE
VSprite	Gels	TYPE
VSpriteFlagSet	Gels	TYPE
VSpriteFlags	Gels	TYPE
VSpritePtr	Gels	TYPE
VUserStuff	Gels	TYPE
View	Views	TYPE
ViewAddress	Windows	PROCEDURE
ViewPort	Views	TYPE
ViewPortAddress	Windows	PROCEDURE
ViewPortPtr	Views	TYPE
ViewPtr	Views	TYPE
VolRange	NarratorDevice	TYPE
WBArg	Workbench	TYPE
WBArgPtr	Workbench	TYPE
WBOObject	Workbench	TYPE
WBOObjectPtr	Workbench	TYPE
WBOObjectType	Workbench	TYPE
WBPortMsgType	Workbench	TYPE
WBStartup	Workbench	TYPE
WBenchToBack	Screens	PROCEDURE
WBenchToFront	Screens	PROCEDURE
Wait	Tasks	PROCEDURE
WaitBOVP	Views	PROCEDURE

WaitBlit	Blitter	PROCEDURE
WaitForChar	DOSFiles	PROCEDURE
WaitIO	IO	PROCEDURE
WaitPort	Ports	PROCEDURE
WaitTOF	Views	PROCEDURE
Window	Intuition	TYPE
WindowFlags	Intuition	TYPE
WindowFlagsSet	Intuition	TYPE
WindowLimits	Windows	PROCEDURE
WindowPtr	Intuition	TYPE
WindowToBack	Windows	PROCEDURE
WindowToFront	Windows	PROCEDURE
Write	Terminal	PROCEDURE
Write	InOut	PROCEDURE
Write	DOSFiles	PROCEDURE
WriteCard	InOut	PROCEDURE
WriteChar	Streams	PROCEDURE
WriteHex	InOut	PROCEDURE
WriteInt	InOut	PROCEDURE
WriteLn	Terminal	PROCEDURE
WriteLn	InOut	PROCEDURE
WriteOct	InOut	PROCEDURE
WritePixel	Pens	PROCEDURE
WriteString	Terminal	PROCEDURE
WriteString	InOut	PROCEDURE
WriteString	Streams	PROCEDURE
WriteWord	Streams	PROCEDURE
XorRectRegion	Regions	PROCEDURE
arctan	MathLib0	PROCEDURE
bltnode	Blitter	TYPE
bltnodeptr	Blitter	TYPE
cListDescriptor	CListLibrary	TYPE
cPoolDescriptor	CListLibrary	TYPE

collTable	Gels	TYPE
collTablePtr	Gels	TYPE
copinit	Copper	TYPE
copinitptr	Copper	TYPE
cos	MathLib0	PROCEDURE
cprrlist	Copper	TYPE
cprrlistptr	Copper	TYPE
entier	MathLib0	PROCEDURE
exp	MathLib0	PROCEDURE
ioFlagSet	IO	TYPE
ln	MathLib0	PROCEDURE
log	MathLib0	PROCEDURE
power	MathLib0	PROCEDURE
real	MathLib0	PROCEDURE
sin	MathLib0	PROCEDURE
sqrt	MathLib0	PROCEDURE
tan	MathLib0	PROCEDURE
termCH	InOut	VAR

Appendix E

E.1 Run time errors

- 3: Arithmetic overflow trap
- 4: Out of range trap
- 5: Division by zero trap
- 7: Address error trap
- 9: Program halt
- 10: No return from a function
- 11: Illegal case index range
- 12: Stack overflow
- 13: Out of range
- 14: Arithmetic overflow
- 15: Not enough workspace for new process
- 16: Process terminated
- 17: Unimplemented routine
- 18: Normal return

E.2 Compiler error codes

- 0: illegal character in source file
- 2: constant out of range
- 3: open comment at end of file
- 4: string terminator not on this line
- 5: too many errors
- 6: string too long
- 7: too many identifiers (identifier table full)
- 8: too many identifiers (hash table full)
- 20: identifier expected
- 21: integer constant expected
- 22:) expected
- 23: ; expected
- 24: block name at the END does not match
- 25: error in block
- 26: := expected
- 27: error in expression
- 28: THEN expected

29: error in LOOP statement
30: constant must not be CARDINAL
31: error in REPEAT statement
32: UNTIL expected
33: error in WHILE statement
34: DO expected
35: error in CASE statement
36: OF expected
37: : expected
38: BEGIN expected
39: error in WITH statement
40: END expected
41:) expected
42: error in constant
43: = expected
44: error in TYPE declaration
45: (expected
46: MODULE expected
47: QUALIFIED expected
48: error in factor
49: error in simple type
50: , expected
51: error in formal type
52: error in statement sequence
53: . expected
54: export at global level not allowed
55: body in definition module not allowed
56: TO expected
57: nested module in definition module not allowed
58: } expected
59: .. expected
60: error in FOR statement
61: IMPORT expected
70: identifier supplied twice in import list
71: identifier not exported from qualifying module
72: identifier declared twice
73: identifier not declared
74: type not declared
75: identifier already declared in module environment

76: dynamic array must not be value parameter
 77: too many nesting levels
 78: value of absolute address must be of type CARDINAL
 79: scope table overflow in compiler
 80: illegal priority
 81: definition module belonging to implementation not found
 82: structure not allowed for implementation of hidden type
 83: procedure implementation different from definition
 84: not all defined procedures or hidden types implemented
 85: name conflict of exported object or enumeration constant in environment
 86: incompatible versions of symbolic modules
 88: function type is not scalar or basic type
 90: pointer-referenced type not declared
 91: tagfield type expected
 92: incompatible type of variant constant
 93: constant used twice
 94: arithmetic error in evaluation of constant expression
 95: incorrect range
 96: range only with scalar types
 97: type-incompatible constructor element
 98: element value out of bounds
 99: set-type identifier expected
 100: declaration needs too much space
 101: undeclared identifier in export list of module
 102: range not belonging to basic type
 103: wrong class of identifier
 104: no such module name found
 105: module name expected
 106: scalar type expected
 107: set too large
 108: type must not be INTEGER or CARDINAL or ADDRESS
 109: scalar or subrange type expected
 110: variant value out of bounds
 111: illegal export from program module
 112: code block for modules not allowed
 120: incompatible types in conversion
 121: this type is not expected
 122: variable expected

123: incorrect constant
124: no procedure found for substitution
125: unsatisfying parameters of substituted procedure
126: set constant out of range
127: error in standard procedure parameters
128: type incompatibility
129: type identifier expected
130: type impossible to index
131: field not belonging to a record variable
132: too many parameters
134: reference not to a variable
135: illegal parameter substitution
136: constant expected
137: expected parameters
138: BOOLEAN type expected
139: scalar types expected
140: operation with incompatible type
141: only global procedure or function allowed in expression
142: incompatible element type
143: type incompatible operands
144: no selectors allowed for procedures
145: only function call allowed in expression
146: arrow not belonging to a pointer variable
147: standard function or procedure must not be assigned
148: constant not allowed as a variant
149: SET type expected
150: illegal substitution to word parameter
151: EXIT only in LOOP
152: RETURN only in PROCEDURE
153: expression expected
154: expression not allowed
155: type of function expected
156: integer constant expected
157: procedure call expected
158: identifier not exported from qualifying module
159: code buffer overflow
160: illegal value for code
161: call of procedure with lower priority not allowed
198: CARDINAL constant expected

199: BITSET type expected
 200: size of structured type too large for this processor
 201: array index too large for this element type
 202: array element size too large for this processor
 203: array index type too large for this processor
 204: subrange too large for this processor
 206: illegal subrange type
 207: case label range too large
 208: global data too large for this processor
 209: local data too large for this processor
 210: parameter data too large for this processor
 211: offset of record field too large for this processor
 300: index out of range
 301: division by zero
 303: CASE label defined twice
 304: this constant is not allowed as case label
 400: expression too complicated (register overflow)
 401: expression too complicated (codetable overflow)
 402: expression too complicated (branch too long)
 403: expression too complicated (jump table overflow)
 404: too many globals, externals and calls
 405: procedure or module body too long (codetable)
 406: expression too complicated (level overflow)
 923: standard procedure or function not implemented
 924: parameter must not be accessed by a WITH
 941: displacement overflow in index addressing mode
 942: 32 bit by 32 bit multiply/divide not yet implemented
 943: index range must not exceed positive integer range
 944: jump too long (overflow in pc-relative offset)
 945: offset too long (overflow in pc-relative offset)
 946: FOR control variable is not of simple addressing mode
 973: DOWNTO only implemented for step -1
 974: step 0 in FOR statement
 981: constant out of legal range
 982: overflow/underflow in range/offset/address calculation
 990: too many WITH nested
 991: CARDINAL divisor too large (> 8000H)
 992: FOR control variable must not have byte size (for step < > -1 or 1)
 993: INC, DEC not implemented with 2 argument for byte variable

994: too many nested procedures
995: FOR step too large (> 7FFFH)
996: CASE label too large (> 7FFFH)
997: type transfer function not implemented
998: FOR limit too large
999: missing symbol file(s)



Modula-2 Features

- Full interface to ROM Kernel, Intuition and AmigaDos.
- 32 bit native code implementation - no code or data limitations.
- Full implementation of Modula-2 with all standard modules.
- Supports transcendental functions and real numbers.
- Separate compilation of modules with version control.
- CODE statement for in-line assembly code.
- Ability to quickly locate and identify errors in source code.
- Modula-2 is NOT copy protected.

Modula-2, designed by Professor Niklaus Wirth (the creator of Pascal) as the successor to Pascal, is a language designed to encourage the user to write in modules. This method of programming makes software easy to design, write, and maintain. Pascal programmers, in particular, will learn the language in a few hours.

Modula-2 is designed to completely replace assembly language programming while providing a more structured, readable and maintainable human interface than "C".

To speed up the development processes, the compiler automatically creates a separate task which runs the error lister. This allows the user to run the text editor in one window while another window displays the errors complete with a descriptive explanation.

One of the unique features of TDI Modula-2 is the full interface to Intuition, AmigaDos and ROM Kernel. Modula-2 can access any of the functions that are available in "C" but in a more elegant and structured fashion.

The full source to several example programs that demonstrate opening multiple windows, color graphics, speech and many other features of the Amiga is included.

TDI Modula-2 comes in two versions: regular and developer's. The developer's version supplies an extra diskette containing all of the definition module sources, a symbol file decoder, link and load file disassemblers, a source file cross referencer, the kermit file transfer utility and the source code to several of the Amiga Modules.

SYSTEM REQUIREMENTS: Amiga with 512K Ram.